LASER™
PERSONAL COMPUTER 110

BASIC
PROGRAMMIERHANDBUCH

VTECH

Die in dieser Dokumentation enthaltenen Informationen unterliegen unangekündigten Änderungen und stellen keine Verpflichtung von seiten der Video Technology Ltd. dar. Das ganze oder teilweise Duplizieren des BASIC Kassettenbandes, der Diskette, des ROM oder irgendeines anderen Mediums für welche Zwecke auch immer ohne schriftliche Genehmigung seitens der Video Technology Ltd. sind untersagt.

ERSTAUSGABE – 1983

EINLEITUNG

Diese Handbuch soll Ihnen helfen, das Programmieren in BASIC zu erlernen. Nach einer Zeit der Einarbeitung werden Sie festellen, daß es nicht sehr schwierig ist, den LASER 110 in BASIC zu programmieren. Sie werden in die Grundlagen der Programmierung eingeführt. Es wird nichts vorausgesetzt, und alle Punkte werden nacheinander Schritt für Schritt erklärt. Beginnen Sie am Anfange dieses Handbuches, und arbeiten Sie die Kapitel in ihrer Reihenfolge durch. Gehen Sie erst weiter, wenn Sie den Inhalt des vorhergehenden Kapitels verstanden haben.

Um Erfolg zu haben, reicht es nicht, dieses Buch nur zu lesen. BASIC lernen Sie am und mit dem LASER 110. Klavierspielen und Schimmen lernen Sie auch nicht nur mit dem Handbuch, sondern durch die Anwendung in der Praxis. Ärgern Sie sich nicht über Fehler. Sie sind Bestandteil des Lernprozesses. Machen Sie einen Fehler, so korrigieren Sie ihn und arbeiten weiter. Der Computer nimmt es Ihnen nicht übel, warum Sie?

Hinweise im Text machen Sie darauf aufmerksam, wenn Anweisungen bereits erarbeitete Daten zerstören können. Ansonsten probieren Sie nach Belieben alles, was Ihnen erklärt wird. Erarbeiten Sie bitte die Kapitel in ihrer Reihenfolge. Nur Kapitel 15 "Programmspeicherung auf Kassette" sollte vor der ersten Abspeicherung auf Kassette erarbeitet werden. Dieses Handbuch ist nicht nur eine Einführung in das BASIC LASER 110, sondern eine grundlegende Einführung in BASIC für alle Systeme. Beachten Sie aber, daß es viele Unterschiede zwischen den einzelnen Systemen gibt.

Wir hoffen, daß dieses Handbuch Ihnen hilft, mit den BASIC-Grundlagen vertraut zu werden.

Wir wünschen Ihnen viel Spaß mit dem
LASER 110 COMPUTER SYSTEM.

INHALT

KAPITEL 1

DER COMPUTER

— WAS IST EIN COMPUTER?
— WORAUS BESTEHT EIN COMPUTER-SYSTEM?
— WAS IST EIN PROGRAMM?
— DIE COMPUTER-SPRACHE
— BASIC

WAS IST EIN COMPUTER?

Ein Computer ist ein Gerät, welches verschiedene Operationen nach einer vom Benutzer erstellten Anweisungsliste ausführt. Er kann nicht selbständig Probleme lösen, aber er löst sie, wenn ihm gesagt wird, wie. Der Computer kann nicht denken, aber er ist in der Hand eines Programmierers ein effektives Werkzeug.

Ein Computersystem besteht aus einer Anzahl von Geräten, deren Arbeit von einer zentralen Kontrolleinheit koordiniert wird. Wenn dieses System aktiv ist, ist es in der Lage, einfache logische Vergleiche und arithmetische Operationen auszuführen. Informationen können eingespeichert werden und verständliche Ergebnisse ausgegeben werden.

WORAUS BESTEHT EIN COMPUTER-SYSTEM?

Ein Computer-System besteht aus folgenden Geräten:

i) Die CPU (central processing unit)
Sie führt die in der Instruktionsliste notierten Anweisungen aus. Es können arithmethische, logische und speicher-bezogene Operationen ausgeführt werden. Man kann die CPU als das Gehirn des Computers bezeichnen.

ii) Die Speicher-Einheit — In ihr werden die Instruktionsliste und alle von der CPU oder vom Benutzer gegebenen Informationen abgelegt. Die Speicher-Einheit befindet sich innerhalb des Computers. Die CPU kann direkt auf alle Instruktionen und Informationen zugreifen.

iii) Der Massen-Speicher — Er befindet sich ausserhalb des Computers und nimmt ebenfalls Instruktionslisten und Informationen von Benutzer oder CPU auf. Tonbandgerät und Floppy-Disk sind Massenspeicher. Alle Informationen und Instruktionen müssen zur Bearbeitung durch die CPU in den Computerspeicher transportiert werden.

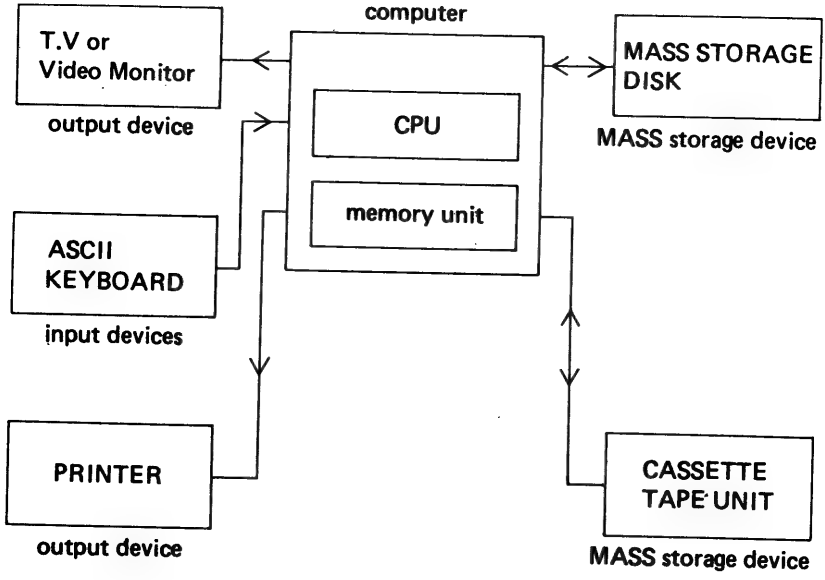
iv) Eingabe-Gerät (Input-Device) — Eingabegeräte geben dem Benutzer die Möglichkeit, Daten oder Instruktionen

in den Computer zu geben. Die Tastatur ist ein Beispiel eines Eingabegerätes.

v) Ausgabe-Geräte (Output Devices) – Sie empfangen von der CPU Informationen und die Ergebnisse der Operationen. Drucker und Monitor sind typische Ausgabegeraete.

Ein- und Ausgabekanal stellen für den Benutzer einen Kommunikationskanal zum Computersystem dar, über den er unter Steuerung der CPU mit dem Computersystem Informationen austauchen kann.

Grösse und Ausbau von Computersystemen können entsprechend den Anforderungen sehr unterschiedlich sein, jedoch sind die beschriebenen Einheiten in allen Systemen vorhanden.



Computer-System, grundsätzlicher Aufbau

WAS IST EIN PROGRAMM?

Ein Programm ist eine Liste von Anweisungen. Der Entwurf solcher Listen wird Programmieren genannt.

Ein Programmierer erstellt also eine Liste. Diese Liste, in den Computer eingegeben, ermöglicht die schrittweise Lösung einer Aufgabe durch den Computer.

DIE COMPUTER-SPRACHE

Um ein Programm für einen Computer zu erstellen, ist in zwei Schritten vorzugehen. Erstens muß der Benutzer wissen, welche Instruktionen er anzuwenden hat und in welcher Form er Sie definieren muß, und zweitens muß er in der Lage sein, die Instruktionsliste dem Computer zu übermitteln. Es findet eine Art der Kommunikation statt: durch das Schreiben des Programmes in einer Programm-Sprache und das Lesen und Ausführen des Programmes durch den Computer.

Es sind sehr viele Programmsprachen in Gebrauch. Teilweise sind sie für spezielle Anwendungen vorgesehen, teilweise decken sie jedoch ein breites Feld von Anwendungen ab. BASIC ist der letzteren Kategorie zuzuordnen.

BASIC

BASIC ist eine Abkürzung: "Beginner's All-purpose Symbolic Instruction Code". BASIC hat ein einfaches, englisches Vokabular und nur wenige grammatikalische Regeln. BASIC folgt einfachen mathematischen Grundsätzen. Um ihrem Computer Instruktionen zu übermitteln, müßen Sie BASIC können. BASIC wird nachfolgend schrittweise erklärt.

Programme, die in BASIC geschrieben sind, werden durch ein Übersetzungs-Programm in die für die CPU verständliche Maschienensprache übersetzt. Dieses Programm wird BASIC-INTERPRETER genannt und befindet sich in Ihrem Gerät.

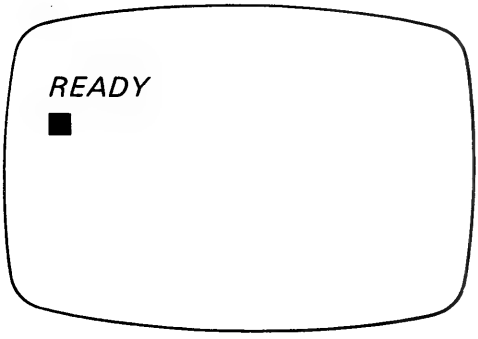
KAPITEL 2

WIE SIE IHREN COMPUTER BENUTZEN

— DER START
— DIE ARBEIT MIT DER TASTATUR
— PRINT UND RETURN
— SYNTAX ERROR
— EDITIEREN
— INSERT
— CLS
— EIN AUSBLICK

DER START

Wenn Sie Ihren Computer angeschlossen und eingeschaltet haben, wird die folgende Meldung auf dem Bildschirm erscheinen:



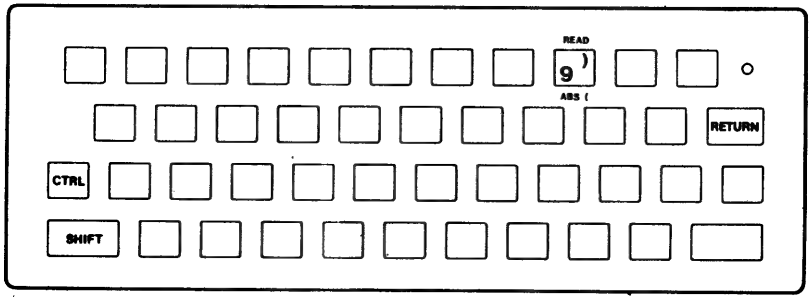
READY zeigt an, daß der Computer im Wartezustand ist und auf Anweisungen wartet. Das blinkende Quadrat ist der CURSOR. Er blinkt stets an der Position, an der das nächste Zeichen Ihrer Eingabe erscheinen wird.

DIE ARBEIT MIT DER TASTATUR

Die Tastatur sieht zwar etwas kompliziert aus, ermöglicht aber ein angenehmes und sicheres Arbeiten.

Nehmen wir die Taste [9] als Beispiel: Um die Zahl [9] einzugeben, drücken Sie einfach die Taste 9. Wollen Sie jedoch die Anweisung **READ** erzeugen, betätigen Sie die Taste [CTRL] und gleichzeitig die Taste [9] . Soll die Klammer eingegeben werden, sind zu gleich [SHIFT] und [9] zu betätigen. Um das Wort **ABS(** unter der Taste zu erzeugen, ist die folgende Tastenkombination zu geben: erst das gleichzeitige Betätigen von [CNTRL] und [RETURN], dann Taste [9].

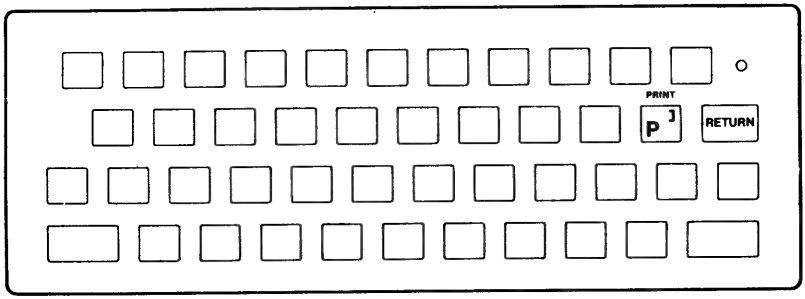
Mit etwas Übung arbeiten Sie mit dieser Tastatur ohne Schwierigkeiten. Probieren Sie alles, was Ihnen einfällt, nach dem Motto "Was passiert, wenn . . . . . " aus, und Sie werden es herausfinden.



PRINT AND RETURN

Sie geben Ihrem Computer Anweisungen in der Programm-Sprache BASIC. Der Computer kann diese Anweisungen sofort ausführen, oder er kann sie speichern und später als Programm abarbeiten.

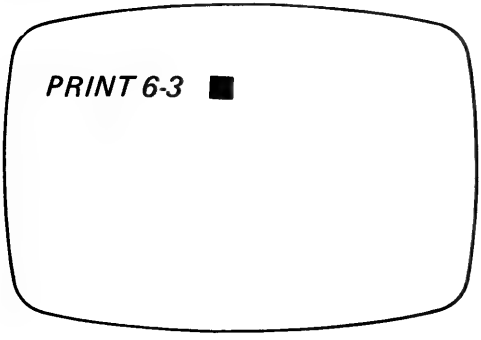
Wir wollen nun dem Computer eine sofort auszuführende Anweisung geben. Dazu brauchen wir zwei Worte der BASIC-Programmsprache:



Zur Eingabe dieser oder anderer Anweisungen können Sie die Worte ausschreiben wie **P, R, I, N, T,** oder Sie können die entsprechende Tastenkombination, in diesem Falle **CNTRL** - **P** benutzen. Probieren Sie es auf Ihrer Tastatur aus.

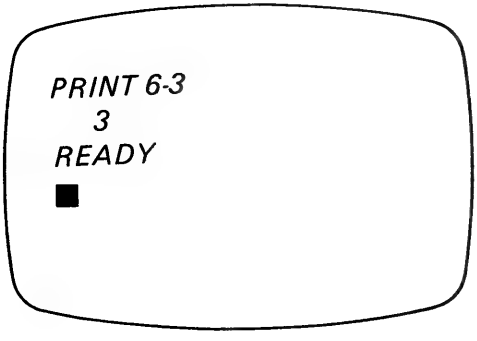
Wenn Sie die Anweisung **PRINT** eingeben, weiß der Computer, daß er die auf **PRINT** folgende Information ausgeben soll.

Schreiben Sie *PRINT 6-3*, und der Bildschirm zeigt Ihre gegebene Anweisung.



Beachten Sie, daß bei jeder Tastenbetätigung ein Kontrollton hörbar wird, der Ihnen anzeigt, daß die Tastenbetätigung registriert wurde. Eine dauernde Kontrolle der Eingaben auf dem Bildschirm kann daher bei entsprechender Übung entfallen.

Betätigen Sie nun die Taste **RETURN** , und der Bildschirm wird folgendes anzeigen:

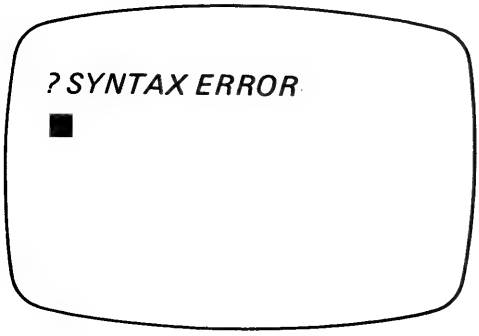


Die Betätigung der **RETURN** -Taste zeigt dem Computer die Beendigung der Eingabe an. Die Eingabezeile wird sofort ausgeführt.

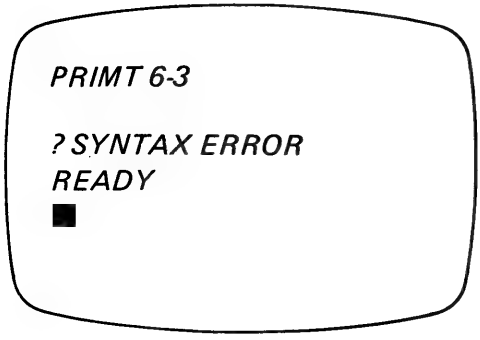
Beachten Sie also, daß **RETURN** jede komplette Eingabe abschließt.

SYNTAX ERROR

Die Bildschirm-Anzeige: ? **SYNTAX ERROR** erscheint stets nach Betätigen der **RETURN** -Taste bei fehlerhafter Schreibweise oder falscher Zeichensetzung.



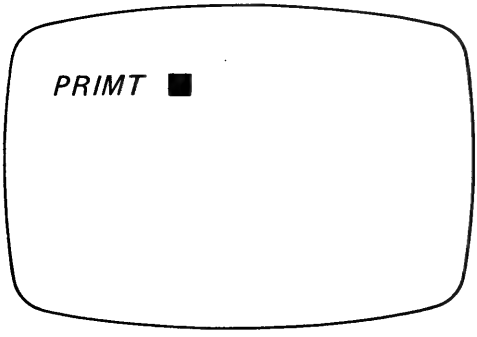
Schreiben Sie beispielsweise statt *PRINT 6-3* jetzt *PRIMT 6-3* und geben dann **RETURN**, erscheint folgendes auf dem Bildschirm:



Außer der Fehlermeldung **SYNTAX ERROR** existieren noch viele andere Fehlermeldungen. Eine Liste aller Fehlermeldungen finden Sie im Anhang.

EDITIEREN

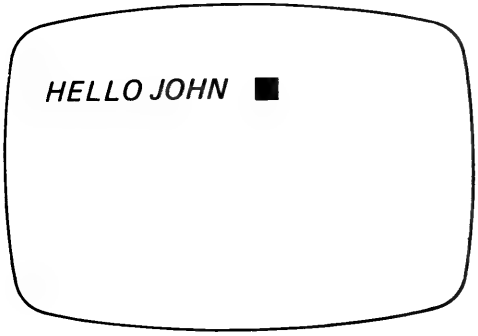
Sollten Sie bei der Eingabe einer Anweisung einen Fehler gemacht haben, so können Sie mit der CNTRL -Taste und weiteren Tasten den Fehler berichtigen:



Blinkt der Cursor über dem **T**, so ist mit CNTRL und dem gleichzeitigen Betätigen von RUBOUT das Zeichen zu löschen. Mit CNTRL ← wird der Cursor über das Zeichen **M** bewegt und CNTRL RUBOUT löschen auch das falsche **M** aus. Mit der Eingabe **N** **T** wird das Wort zu **PRINT** berichtigt.

Der Cursor läßt sich in vier Richtungen bewegen: aufwärts, abwärts, links und rechts. Die Bewegungsrichtung wird durch die Pfeilmarkierungen über den Tasten rechts unten auf der Tastatur angegeben.

Ein Beispiel: Wenn Sie einige Worte geschrieben haben, befindet sich der Cursor rechts neben dem letzten Zeichen. Sie wollen ein Wort löschen: Drücken Sie **CNTRL** und ← so lange, bis der Cursor über dem ersten Zeichen des zu löschenden Wortes steht. Mit **CNTRL** **RUBOUT** löschen Sie das Wort.



Soll das Wort **JOHN** gelöscht werden, so wird also mit **CNTRL** [←] der Cursor über das **J** gebracht und mit **CNTRL** **RUBOUT** das Wort gelöscht. Probieren Sie es bitte!

Es ist ebenfalls möglich, Zeichen zu überschreiben. Bringen Sie den Cursor über das **O** und schreiben Sie **AMES**, und Sie haben das Wort **JAMES** auf dem Bildschirm.

Sollte der Cursor nicht mehr in der Zeile stehen, in der Sie ein Wort ändern wollen, so können Sie ihn mit den Tasten **CNTRL** [↑] zeilenweise aufwärts bewegen. Haben Sie die entsprechende Zeile erreicht, so wird wie im vorstehenden Beispiel das Editieren vorgenommen.

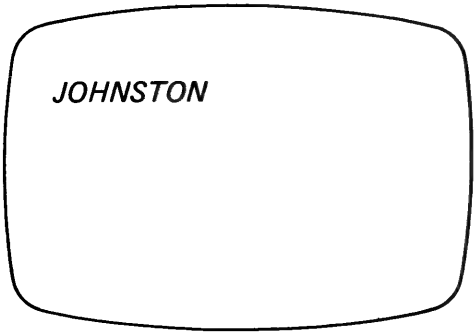
Machen Sie sich nach dem Motto "Übung macht den Meister!" mit den Editier-Funktionen vertraut.

INSERT

Diese Tastenfunktion erlaubt das Einfügen von Zeichen ab der Cursor-Position, ohne den bereits geschriebenen Text weiter zu ändern. Als Beispiel: In das Wort **JOHNTON** ist zwischen **N** und **T** ein **S** einzufügen.

Der Cursor ist dazu auf das **N** zu bringen, und mit den Tasten **CNTRL** **L** fügt die Insert-Funktion einen Leerraum ein, in den das Zeichen **S** geschrieben werden kann.

Der Bildschirm zeigt nun:



Nach dem Editieren einer Zeile ist auf jeden Fall die **RETURN** -Taste zu betätigen. Die Änderungen werden sonst nicht als Eingabe vom Computer übernommen, und die fehlerhafte Zeile kommt zur Ausführung.

CLS

Sie haben inzwischen festgestellt, das CTRL für control steht.

Mit der Tastenkombination CNTRL H bringen Sie die Anweisung CLS (clear screen) auf den Bildschirm und mit RETURN zur Ausführung. Der Bildschirm wird gelöscht, nicht jedoch die vorher mit RETURN in den Speicher übernommenen Anweisungen.

Mit CNTRL 8 bringen Sie das Wort **NEW** auf den Bildschirm. RETURN bringt NEW zur Ausführung, und Bildschirm und Speicher(!) werden gelöscht.

Der Speicher wird ebenfalls bei Ausfall der Stromversorgung oder Ausschalten des Computers gelöscht!

EIN AUSBLICK

An diesem Punkt der Einarbeitung in die BASIC-Programmierung können wir einen Ausblick auf weiterführende Punkte machen. Benutzen Sie Ihre bisher erworbenen Kenntnisse zur Eingabe des folgenden Programmes.

Geben Sie es sorgfältig, ohne Rücksicht auf die Bedeutung der Anweisungen, ein.

```
10 FOR I = Ø TO 1 RETURN
20 FOR J = Ø TO 255 RETURN
30 POKE 28672 + I*256 + J, J RETURN
40 NEXT RETURN
50 NEXT RETURN
60 GOTO 60 RETURN
   RUN RETURN
```

Alle darstellbaren Zeichen erscheinen auf dem Bildschirm. Mit CNTRL BREAK können Sie wieder in die Eingabe-Betriebsart mit dem blinkenden Cursor zurückschalten.

KAPITEL 3

IHR COMPUTER ALS EINFACHER RECHNER

— EINFACHE ANWEISUNGEN
— DIE REIHENFOLGE NUMERISCHER OPERATIONEN
— KLAMMERN

EINFACHE ANWEISUNGEN

Um den Computer als einfachen Rechner zu benutzen, braucht nur **PRINT**, gefolgt von der Problemlösung und dann [RETURN] eingegeben werden.

Der Computer addiert mit [+], subtrahiert mit [–], multipliziert mit [*], dividiert mit [/] und errechnet eine Zahl zur Basis x mit [↑]. Die Anweisungszeichen leiten OPERATIONEN ein, die Zahlen werden als OPERANDEN bezeichnet.

Schreiben Sie als Beispiel

```
PRINT 3 ↑ 2 [RETURN]
```

und der Computer wird mit 9 antworten.

DIE REIHENFOLGE NUMERISCHER OPERATIONEN

Zusammengesetzte Rechenoperationen führt der Computer in einer festgelegten Reihenfolge, die von der Schreibweise abweicht, aus. Die Reihenfolge:

1) Minuszeichen als Vorzeichen negativer Zahlen.
2) Exponentation, links beginnend, nach rechts.
3) Multiplikation und Division gleichwertig, auch hier von links nach rechts.
4) Subtraktion und Addition, gleichwertig und von links nach rechts.

Beispiel A:

```
PRINT 3 ↑ 2 ↑ 2 ↑ 2 [RETURN]
6561
```

Der Computer rechnet 3 ↑ 2 = 9, dann 9 ↑ 2 = 81 und dann 81 ↑ 2 = 6561

Beispiel B:

```
PRINT 6 * 2 + 3  RETURN
15
```

Hier errechnet der Computer erst 6 * 2 und addiert dann 3.

Beispiel C:

```
PRINT 6 + 3 * 4 + 6/3  RETURN
20
```

Erst werden Multiplikation und Division ausgeführt, dann wird 6 + 12 + 2 errechnet.

KLAMMERN

Alle Rechenoperationen in Klammern werden vor anderen Operationen ausgeführt.

Beispiel:

```
PRINT 18/(3 + 3)  RETURN
3
```

Werden Klammerausdrücke geschachtelt, so werden stets die inneren Klammern zuerst aufgelöst.

Beispiel:

```
PRINT 20/(1 + (3 ↑ 2))  RETURN
2
```

KAPITEL 4

KONSTANTEN UND VARIABLEN

— KONSTANTEN
— VARIABLEN
— LET
— SEMIKOLONS UND KOMMAS
— DOPPELPUNKT

KONSTANTEN

In den vorigen Kapiteln haben wir bereits Konstanten gebraucht. Konstanten sind Werte, welche sich nicht ändern und als Zahlen positiv oder negativ werden können. Die Zahl 6,32 ist eine Konstante, 6,33 aber eine andere.

Der Bereich der Zahlen $-10^{38} \leqslant x \leqslant 10^{38}$.

Die kleinste positive Zahl ist 10^{-38}.

VARIABLEN

Als Variable wird ein Wert bezeichnet, der sich ändern kann. Er hat einen Namen: Y = x + 3. Hier wird der Variablen Y der Wert der Variable X plus 3 zugewiesen, wobei X und Y Variablennamen sind. Ein Name kann jeder Buchstabe des Alphabetes sein, jede Kombination von zwei Buchstaben oder jede Kombination aus Buchstabe und Ziffer Ø bis 9. Das erste Zeichen eines Namens muß ein Buchstabe sein. Zum Beispiel: A, AB, A6.

Der Name kann beliebig viele Zeichen haben, wobei nur die beiden ersten Zeichen gewertet werden. Die Variable HALLO ist die gleiche wie HA. Ein Variablenname kann kein BASIC-Wort wie LET oder PRINT sein.

LET

LET weist einer Variablen einen Wert zu. Eine Variable, der kein Wert zugewiesen worden ist, hat einen Wert gleich Null. Die Variable behält ihren Wert, bis er durch Anweisungen wie **LET, READ** und **INPUT** geändert wird.

Beispiel:

```
LET A = 7   RETURN
LET B = 9   RETURN
PRINT A + B RETURN
16
```

Das = -Zeichen in BASIC entspricht nicht dem mathematischen Zeichen, das den Wert links des = gleich den rechten Wert setzt. BASIC spricht von einer Wertzuweisung: Der Variablen links des = wird der Wert des Ausdrucks rechts des = zugewiesen. Links steht also immer ein Variablenname.

Beispiele:

```
LET A = 2  RETURN
LET B = 3  RETURN
LET C = 20  RETURN
LET A = 2 + A  RETURN
LET D = 3 + D  RETURN
PRINT A; B; C; D  RETURN
4  3  20  3
```

In der 4.Reihe wird A der alte Wert plus 2 zugewiesen. A wir also der Wert 4 zugewiesen. In der 5.Reihe wird D der alte Wert plus 3 zugewiesen. Da D vorher keine Zuweisung erhalten hatte, wird D = Ø + 3.

Für Ihren Computer ist der Gebrauch von **LET** nicht zwingend. Eine Zuweisung wird schon durch das = -Zeichen vorgenommen. **LET** A = 7 kann auch als A = 7 notiert werden.

SEMIKOLONS UND KOMMAS

Werden mehrere Variablen in einer **PRINT**-Anweisung ausgeben, so müßen sie durch (,) oder (;) getrennt werden. Beachten Sie den Gebrauch des (;) in vostehendem Beispiel. Es bewirkt, daß alle Resultate in einer Zeile, nur getrennt durch ein Leerzeichen, ausgegeben werden. Die Ausgabe von Zeichenketten wird bei Trennung durch (;) ohne Leerzeichen aneinandergereiht.

Ein Komma anstelle des Semikolons tabuliert die Ausgabe in zwei Kolonnen. Die 32 Positionen des Bildschirms sind als zwei Blöcke mit je 16 Positionen dargestellt. Die erste Ausgabe erfolgt am Beginn des ersten Blockes, die zweite in der gleichen Zeile am Beginn des zweiten Blockes und die dritte in der nächsten Zeile mit Beginn des ersten Blockes. Ist ein Ergebnis länger als 16 Zeichen, so wird der folgende Block benutzt, und die nächste Ausgabe beginnt mit dem darauf folgenden Block.

DOPPELPUNKTE

Werden in einer Zeile mehrere Anweisungen gegeben, so sind sie durch Doppelpunkte (:) zu trennen.

Beispiel:

```
10 FOR I = 1 TO 5 : PRINT I; : NEXT
RUN
1  2  3  4  5
```

LISTE MIT PRINT-ANWEISUNGEN

Beispiel:

```
10 PRINT 4 RETURN
20 PRINT 6 RETURN
30 PRINT 8 RETURN
RUN RETURN
4
6
8
```

SEMIKOLONS AM ENDE DER PRINT-ANWEISUNGEN

```
10 PRINT 4; RETURN
20 PRINT 5; RETURN
30 PRINT 6; RETURN
RUN RETURN
4  5  6
```

KOMMAS AM ENDE DER PRINT-ANWEISUNGEN

```
10 PRINT 4, RETURN
20 PRINT 5, RETURN
30 PRINT 6, RETURN
RUN RETURN
4                    5
6
```

KAPITEL 5

PROGRAMMIEREN

— INPUT
— REM
— NEW
— RUN
— LIST
— PAUSE IM LISTING
— LÖSCHEN EINER ANWEISUNGSZEILE

Lassen Sie uns nun einfache Programme versuchen. In den letzten Kapiteln haben wir Anweisungen zur "direkten Ausführung" gegeben. Nun wollen wir Anweisungen im Computer speichern und sie später zur Ausführung bringen.

INPUT

Sehen Sie sich das folgende Programm an!

```
1Ø   REM ZAHL ZUR BASIS 3
2Ø   INPUT A
3Ø   PRINT A;A ↑ 3
```

Die **INPUT**-Anweisung in Zeile 2Ø verlangt die Zuweisung eines Wertes an die Variable A von der Tastatur. Wird das Programm mit **RUN** zur Ausführung gebracht, so erscheint ein (?) auf dem Bildschirm, und der Programmlauf wird unterbrochen, bis von der Tastatur ein Wert eingegeben und die Eingabe mit **RETURN** abgeschlossen wird.

Beachten Sie, daß jede Zeile mit einer Nummer beginnt. Mit dieser Nummer erkennt der Computer die zu speichernde Anweisung. Die Zeilen werden in der Reihenfolge ihrer Zeilennummern abgespeichert, zur Ausführung gebracht und auf dem Bildschirm dargestellt. Die Numerierung sollte in Zehnerabständen erfolgen. Es können dann später weitere Anweisungen mit dazwischen liegenden Nummern eingeschoben werden. Der Bereich der Zeilennummer ist Ø bis 65529.

REM

Diese Anweisung (Zeile 1Ø) dient zur Einfügung von Notizen in das Programm. Während der Programmausführung ignoriert der Computer Zeilen, die mit einer **REM**-Anweisung beginnen.

Nach einer **REM**-Anweisung dürfen in der gleichen Zeile keine weiteren Anweisungen gegeben werden. Sie kommen nicht zur Ausführung.

Da diese Anweisungen Speicherplatz belegen, können sie bei Speicherplatz-Problemen gelöscht werden.

NEW und RUN

Nun geben wir noch einmal das Programm in den Computer ein. Zuerst jedoch löschen wir mit **NEW** **RETURN** den Speicher und stellen alle Variablen auf Ø. Dann die Eingabe:

```
1Ø REM ZAHL ZUR BASIS 3      RETURN
2Ø INPUT A                   RETURN
3Ø PRINT A; A ↑ 3            RETURN
```

Mit **RUN** **RETURN** kann das Programm nun gestartet werden. Das Fragezeichen dar **INPUT**-Anweisung erscheint, und der Computer erwartet die Eingabe Ihrer Zahl und weist sie der Variable A zu.

Schreiben Sie jetzt 2 **RETURN**

Der Bildschirm zeigt jetzt:

```
1Ø REM ZAHL ZUR BASIS 3
2Ø INPUT A
3Ø PRINT A; A ↑ 3
RUN
? 2
2  8
```

RUN

Nach Eingabe von RUN RETURN wird das gesamte Programm ausgeführt. RUN, gefolgt von einer Zeilennummer mit RETURN , startet das Programm an der durch die Zeilennummer genannten Stelle.

Beachten Sie das folgende Programm:

```
1Ø  INPUT A, B      RETURN
2Ø  PRINT A + B     RETURN
RUN                 RETURN
```

In der ersten Zeile nach **RUN** erscheint das Fragezeichen (?) zur Eingabe eines Wertes für A. In der zweiten Zeile wird mit (??) der Wert für die Variable B gefordert.

Der Bildschirm zeigt:

```
1Ø  INPUT A, B      RETURN
2Ø  PRINT A + B     RETURN
RUN                 RETURN
?     3
??    6
      9
```

LIST

Mit LIST und RETURN wird das gesamte Programm in ansteigender Folge der Zeilennummern auf den Bildschirm gebracht.

Beispiel:

```
10 INPUT A RETURN
20 INPUT B RETURN
30 PRINT A; B; C; A + B + C RETURN
25 INPUT C RETURN
```

```
LIST RETURN
10 INPUT A
20 INPUT B
25 INPUT C
30 PRINT A; B; C; A + B + C
```

Soll nur eine bestimmte Zeile angezeigt werden, so schreiben Sie:

```
LIST (Zeilennummer) RETURN
```

Beispiel:

```
LIST 20 RETURN
20 INPUT B
```

Um einen Teil des Programmes auf dem Bildschirm zu listen, zum Beispiel die Zeilen 2Ø bis 3Ø, schreiben Sie:

```
LIST 2Ø – 3Ø  RETURN
2Ø INPUT B
25 INPUT C
3Ø PRINT A; B; C; A + B + C
```

Wenn Sie *LIST – 3Ø* schreiben, so wird das Programm vom Anfang bis Zeile 3Ø ausgegeben.

Mit *LIST 3Ø –* wird ab Zeile 3Ø bis zum Ende des Programmes die Anweisungsliste auf den Bildschirm ausgegeben.

PAUSE IM LISTING

Um ein mit **LIST** auf den Bildschirm gebrachtes, sehr langes Programmlisting an beliebiger Stelle zu stoppen, muß die Taste **SPACE** (Leertaste) betätigt werden.

LÖSCHEN EINER ANWEISUNGSZEILE

Eine Anweisungszeile im Programm kann durch Eingabe ihrer Nummer, gefolgt von **RETURN**, gelöscht werden.

KAPITEL 6

NUMERISCHE FUNKTIONEN

— WAS IST EINE FUNKTION?

ABS
SGN
SQR
LOG
EXP
INT
RND
SIN
COS
TAN
ATN

WAS IST EINE FUNKTION?

Eine Funktion ist eine Vorschrift, welche auf einen Wert angewandt, einen neuen Wert ergibt. Der erste Wert ist das Argument, der zweite das Resultat.

SQR ist die Wurzel-Funktion. Wenn Sie schreiben:

```
PRINT SQR (9)
```

wird der Computer mit 3 antworten.

In diesem Beispiel ist 9 das Argument, **SQR** die Funktion und 3 das Resultat.

Nachfolgend finden Sie eine Liste der numerischen Funktionen mit einer Kurzbeschreibung. Funktionen, von denen wir annehmen, daß sie für den Leser neu sind, werden anschließend etwas eingehender dargestellt. Beispiele werden nicht gegeben, da alle Funktionen in den Beispielprogrammen der folgenden Kapitel vorkommen.

EINE LISTE DER NUMERISCHEN FUNKTIONEN

Funktion	Beschreibung
ABS (X)	Ergibt den absoluten (positiven) Wert von X.
SGN (X)	Signum-Funktion. Ergibt – 1, wenn X negativ ist. Ergibt + 1, wenn X positiv ist. Ergibt Ø, wenn X gleich Ø ist.
SQR (X)	Ergibt die Wurzel von X. X darf nicht negativ werden.
LOG (X)	Ergibt den narürlichen Logarithmus von X. Das Argument muß größer Null sein.
EXP (X)	Ergibt den Wert von e zur Basis x. e = 2,71828

INT (X)	Ergibt den höchsten, ganzzahligen Wert, der kleiner oder gleich X ist.
RND (X)	Ergibt eine Zufallszahl zwischen 1 und X. RND (Ø) ergibt eine Zufallszahl zwischen Ø = 1.
SIN (X) **COS (X)** **TAN (X)**	Das Argument aller trigonometrischen Funktionen wird als Bogenmaß (rad) notiert. Der Bereich von X: –9999999 bis 9999999.
ATN (X)	Ergibt das Resultat des ARCUS TANGENS in RADIANS.

BEISPIELE ZU ABS, SGN, INT und RND

ABS (X)

Diese Funktion ergibt den absoluten, positiven Wert des Arguments. ABS (–7) is also 7.

Beispiel:

```
PRINT ABS (7 – 2 * 4)  RETURN
1
```

SGN (X)

Diese Funktion ergibt als Resultat +1, wenn X positiv ist, Ø, wenn X Null ist und –1, wenn X negativ ist.

Beispiel:

```
A = –6  RETURN
PRINT SGN (A); SGN (A – A)  RETURN
–1      Ø
```

INT (X)

Diese Funktion wandelt Argumente in die nächst kleinere ganze Zahl unter dem Argument. INT (5.9) ist 5 und INT (–5.9) ist –6.

Beispiel:

```
PRINT INT (–6.7)  RETURN
–7
```

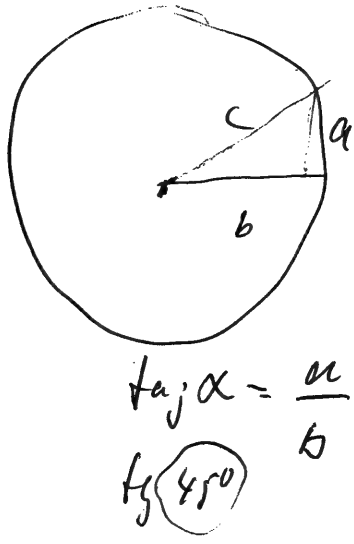
RND (X)

Diese Funktion ergibt eine Zufallszahl zwischen 1 und X. X muß positiv sein.

Beispiel:

```
PRINT RND (19)  RETURN
```

Das Ergebnis ist eine Zahl zwischen 1 und 19.
RND (Ø) wird eine Zahl zwischen Ø und 1 ergeben.
X darf nicht negativ sein.



KAPITEL 7

ZEICHENKETTEN (STRINGS)

— STRING VARIABLEN
— STRING FUNKTIONEN
  LEN
  STR$
  VAL
  LEFT$
  RIGHT$
  MID$
  ASC
  CHR$
— STRING VERGLEICHE
— INKEY$

Sie sind jetzt sicherlich mit dem Gebrauch der **RETURN** – Taste vertraut, und wir werden Sie weiterhin nicht mehr daran erinnern.

Ein String ist eine Kette von Zeichen, die als eine Einheit aufgefaßt werden.

Zeichenketten werden in Anführungszeichen geschrieben.

Beispiel:

```
"HILFE"
```

Werden Zeichenketten mit **PRINT**-Anweisungen ausgegeben, veranlaßt das Semikolon die Aneinanderreihung ohne Leerzeichen.

STRING VARIABLEN

Jeder Buchstabe des Alphabetes kann als Variablenname benutzt werden, jedoch ist ein $ -Zeichen nachzustellen.

Beispiel:

```
A$ = "EIN DUTZEND EIER"
```

Namen können auch zwei Buchstaben oder ein Buchstabe, gefolgt von einer Ziffer Ø –9, sein. Beispiel: A$, AB$, A7$.

Strings können aneinandergereiht (verkettet) werden. Lassen Sie uns das in einem Programm demonstrieren:

```
10 A$ = "ICH BI"
20 B$ = "N 15 JA"
30 C$ = "HRE ALT"
40 PRINT A$ + B$ + C$
   RUN
   ICH BIN 15 JAHRE ALT
```

Beachten Sie hier ebenfalls die Leerzeichen!

STRING FUNKTIONEN

Ihr Computer stellt Funktionen zur Manipulation von Zeichenketten bereit.

LEN

Diese Funktion liefert die Anzahl der Zeichen eines Strings. *PRINT LEN ("JOHN")* ergibt als Resultat 4. Das bedeutet: die Zeichenkette besteht aus 4 Zeichen (Character).

Leerzeichen werden ebenfalls als Character gezählt. Die Kett **"JOHN"** ergibt mit der LEN-Funktion 7.

STR$

Die **STR$**-Funktion wandelt eine Zahl zu einer Kette Ziffern. Beachten Sie die folgende Beispiele:

```
A$ = STR$ (73)
```

hat die selbe Bedeutung, wie:

```
A$ = "73"
```

Beispiel:

```
10 A$ = STR$ (7 * 3)
20 B$ = A$ + "GROSS"
30 PRINT B$
   RUN
   21 GROSS
```

VAL

VAL ist die Umkehrfunktion zu **STR$**. Mit **VAL** wird eine aus Ziffern bestehende Zeichenkette zu einer Zahl gewandelt. **VAL** wandelt nur Ziffern bis zum ersten nichtnumerischen Zeichen der Kette.

```
VAL ("61") = 61
```

Beispiel:

```
10 A$ = "33"
20 B$ = "20"
30 C  = VAL (A$ + B$)
40 PRINT C; C + 100
RUN
3320   3420
```

TEILSTRINGS

Der Substring ist Teil des Strings. Beispielsweise ist "ASD" Teil des Strings "A8DCE".

Es sind in Ihrem Computer Funktionen vorhanden, die das Arbeiten mit Teilen von Zeichenketten ermöglichen.

LEFT$ (A$, N)

Diese Funktion erzeugt einen Teilstring aus **A$**, beginnend mit dem linken Zeichen der Kette bis zum N-Zeichen.

Beispiel:

```
10 A$ = "ABCDE"
20 B$ = LEFT$ (A$ + "FGH", 6)
30 PRINT B$
   RUN
   ABCDEF
```

RIGHT$ (R$, N)

Diese String-Funktion liefert als Resultat einen Teilstring von **A$**. Er besteht aus N-Zeichen und wird rechts von A$ abgekettet.

Beispiel:

```
10 A$ = "WARUM"
20 B$ = RIGHT$ (A$ + "ICH", 4)
30 PRINT B$
   RUN
   MICH
```

MID$ (A$, M, N)

Diese Funktion ergibt einen Teilstring aus **A$**, beginnend bei dem mit **M** spezifierten Zeichen und N-Zeichen lang.

Beispiel:

```
10 A$ = "ABCDEFGH"
20 B$ = MID$ (A$, 2, 3)
30 PRINT B$
   RUN
   BCD
```

ASC

Die **ASC**-Funktion in der Schreibweise **ASC (A$)** ergibt als Resultat den ASCII-Code (Zeichen-Code) des ersten Zeichens in der Zeichenkette. Beispielsweise ist der dezimale Code-Wert des Zeichens "X" gleich 88. Wenn *A$ = "XAB"* ist, dann ergibt ASC (A$) das Resultat 88.

Beispiel:

```
10 X = ASC ("ROY")
20 PRINT X
   RUN
   82
```

CHR$

Diese Funktion arbeitet umgekehrt zur **ASC**-Funktion. Die **CHR$**-Funktion ergibt als Resultat das Zeichen des ASCII-Codes im Argument. Das Argument kann eine Zahl Ø – 255 oder ein numerischer Ausdruck sein. Das Argument wird in Klammern geschrieben.

Beispiel:

```
30 PRINT CHR$ (68)
RUN
D
```

Verhältnis-Testoperatoren (wie auf Seite 73 beschrieben) können auch für den Vergleich von Zeichenketten angewendet werden. Verglichen wird der **ASCII**-Wert jedes einzelnen Zeichens der Ketten. Um beispielsweise Gleichheit festzustellen, muß jedes Zeichen der beiden Ketten (auch die Leerzeichen!) gleich sein.

Beispiel:

```
10 A$ = "AA"
20 B$ = "BA"
30 IF A$ = B$ then PRINT 20
40 IF A$ < B$ then PRINT 30
50 IF A$ > B$ then PRINT 40
   RUN
   30
```

Die ASCII-Werte der Zeichen können Sie einer Tabelle im Anhang entnehmen. In diesem Programm werden A = 65 und B = 66 miteinander verglichen. A mit 65 ist kleiner als B mit 66, und Zeile 40 des Beispiels gibt die Zahl 30 aus.

Wenn die ersten beiden Zeichen gleich sind, werden die nächsten Zeichen getestet, bis beide Zeichenketten bearbeitet sind.

Beispiel:

```
10 A$ = "ABC"
20 B$ = "ABD"
30 IF B$ > A$ then PRINT 40
   RUN
   40
```

Der Vergleich der letzten Zeichen beider Ketten, **C** und **D**, führt zur Aussage: **B$** ist größer als **A$**, und 40 wird auf den Bildschirm ausgegeben.

INKEY$

Mit **INKEY$** wird ein Zeichen von der Tastatur in eine String-Variable eingelesen. Die Variable enthält entweder kein Zeichen (Null-String), wenn keine Taste betätigt wurde, oder ein Zeichen entsprechend der betätigten Taste. Mit **INKEY$** können alle darstellbaren Zeichen eingelesen werden. Die Ausnahme: **CNTRL** **BREAK**, welches den Programmlauf abbricht.

Beispiel:

```
10 A$ = INKEY$
20 PRINT A$;
30 GOTO 10
```

KAPITEL 8

ÜBERARBEITEN DES PROGRAMMES

— GOTO
— BREAK
— CONT
— STOP
— END
— CLEAR

Hier folgt nun die Erklärung einiger Anweisungen, die das Entwerfen interessanterer Programme erlauben.

GOTO

Der Computer bearbeitet die Anweisungen in der Reihenfolge der Zeilennummern. Die GOTO-Anweisung veranlaßt einen Sprung zu der nach GOTO definierten Zeilennummer. Die Programmabarbeitung wird dort fortgesetzt.

Beispiel:

```
10 INPUT A
20 PRINT A, A ↑ 3
30 GOTO 10
   RUN
   ?
```

Weissen Sie jetzt eine 2 der Variablen A zu, so erhalten Sie das Ergebnis 2 und 8, und ein neues Fragezeichen fordert zu einer weiteren Eingabe in A auf. Dieses Verhalten ist das Ergebnis der Anweisung GOTO 10, die die Programmabarbeitung immer wieder neu startet.

BREAK

Sollten Sie keine Lust mehr haben, weiter Zahlen zur Basis 3 zu errechnen, können Sie mit **CNTRL** **BREAK** nach einem (?) das Programm unterbrechen. Der Computer unterbricht die Programmabarbeitung und gibt die Meldung *BREAK IN 10* aus. Beachten Sie, das **BREAK** keine **BASIC**-Anweisung ist. Sie können mit der Tastenfunktion **BREAK** jedes laufende Programm unterbrechen.

CONT

Sollten Sie, nachdem Sie das Programme mit CNTRL BREAK unterbrochen haben, noch weitere Zahlen zur Basis 3 errechnen wollen, können Sie das Programm mit der Anweisung **CONT** wieder fortsetzen.

STOP

Eine wichtige Anweisung für die Programmentwicklung ist die **STOP**-Anweisung. Im Programm erlaubt sie einen Halt an der Stelle, an der sie steht, und ermöglicht eine Inspektion der Variablen. Die Anweisung ist also ein wichtiges Werkzeug zur Fehlersuche.

Danach wird der Programmlauf durch Eingabe von **CONT** bei der auf die **STOP**-Anweisung folgenden Zeile fortgesetzt.

END

Die **END**-Anweisung beendet die Ausführung eines Programmes. Anders als mit der **STOP**-Anweisung, ist kein Fortsetzen des Programmes mit **CONT** möglich.

Beispiel:

```
10 INPUT A
20 IF A > 0 THEN PRINT "A IST POSITIV":
   END
30 IF A < 0 THEN PRINT "A IST NEGATIV":
   END
40 PRINT "A IST NULL"
50 END
```

Beachten Sie, daß die **STOP**-Anweisung ein **BREAK IN LINE XX** ausgibt, die **END**-Anweisung jedoch nicht.

CLEAR

Mit der **CLEAR**-Anweisung kann zur Speicherung von Zeichenketten mehr Speicherraum bereitgestellt werden.

Beispiel:

```
10 CLEAR 1000
```

Die auf **CLEAR** folgende Zahl gibt den für Strings reservierten Speicherbereich an. Es werden also 1000 Bytes zur Speicherung von Stringvariablen definiert. Wird diese Anweisung nicht gegeben, setzt der Computer automatisch einen Wert fest (Dafault-Wert). Mit gröberem Stringspeicher verringert sich jedoch der für die Programmspeicherung zur Verfügung stehende Speicherraum.

KAPITEL 9

VERGLEICHS-OPERATIONEN

— IF. . .THEN. . .ELSE
— BEDINGTE VERZWEIGUNGEN
— LOGISCHE OPERATOREN
— WAHRHEITS-TABELLEN

IF . . . THEN . . . ELSE

Auf dem Weg durch das **BASIC** haben wir immer mehr Möglichkeiten der Programmiersprache erarbeitet. Wir können schon viel mit dem Computer erreichen.

In diesem Kapitel werden wir die **IF . . . THEN . . . ELSE**-Anweisung kennenlernen. Sie ist eine der wichtigsten Anweisungen des **BASIC**-Konzeptes. Eine weitere wichtige Anweisung ist **FOR . . . TO . . . NEXT**. Wir werden sie im nächsten Kapitel kennenlernen.

Sehen Sie sich jetzt dieses Beispiel an:

```
60 IF A$ > B$ THEN PRINT A$ ELSE PRINT B$
```

Diese Anweisung veranlaßt den Computer, wenn **A$** größer als **B$** ist, *PRINT A$* auszuführen, sonst *PRINT B$* auszuführen.

BEDINGTE VERZWEIGUNGEN

Eine Bedingung besteht aus: einem Ausdruck, einem Verhältnis-Operator und einem zweiten Ausdruck. Jeder **BASIC**-Ausdruck kann benutzt werden, jedoch müssen beide Ausdrücke vom gleichen Typ sein. Also beide numerisch oder beide müssen Stringausdrücke sein.

Verhältnisoperatoren bei **IF . . . THEN**-Anweisungen sind:

= Gleich
<= Kleiner oder gleich
<> Ungleich
>= Größer oder gleich
< Kleiner als
> Größer als

Die folgenden Beispiele zeigen die Anwendung dieser Bedingungs-Operatoren in **IF . . . THEN . . . ELSE**-Anweisungen.

IF . . . . THEN A = B
IF . . . . THEN GOTO
IF . . . . THEN GOSUB
IF . . . . THEN PRINT
IF . . . . THEN INPUT

Beispiel 1:

```
30 IF X > 25 THEN 60
```

Wenn die Anweisung nicht wahr ist, d.h. **X** ist nicht größer als 25, wird der Programmablauf mit der nächsten Anweisung fortgesetzt. Die Anwendung des **ELSE**-Teiles ist nicht notwendig, **ELSE** darf weggelassen werden.

Beispiel 2:

```
10 INPUT A, B
20 IF A > B THEN 50
30 IF A < B THEN 60
40 IF A = B THEN 70
50 PRINT A; "IST GROESSER ALS"; B: END
60 PRINT A; "IST KLEINER ALS"; B: END
70 PRINT A; "IST GLEICH"; B
80 END
   RUN
    ?     7
   ??     3
   IST GROESSER ALS 3
```

Beispiel 3:

```
40 IF P = 6 THEN PRINT "WAHR" ELSE
PRINT "UNWAHR"
```

In diesem Beispiel wird der Computer, wenn **P** gleich 6 ist, **WAHR** ausgeben, bei jedem anderen Wert für **P** jedoch **UNWAHR**. Die Programmausführung wird auf jeden Fall in der folgenden Anweisungszeile fortgesetzt.

Es können nach **THEN** und **ELSE** mehrere, durch (:) getrennte Anweisungen folgen.

Beispiel 4:

```
50 IF A = 5 THEN PRINT "WAHR": S = S – 3:
GOTO 90 ELSE PRINT "UNWAHR": K = K + 8
```

Wenn **A** gleich 5 ist, wird der Computer "WAHR" ausgeben, von S 3 subtrahieren und zur Anweisungszeile 90 gehen. Ist **A** nicht 5, so wird er "UNWAHR" ausgeben und zur Variable **K** 8 addieren.

LOGISCHE OPERATOREN

Logische Operatoren werden in **IF . . . THEN . . . ELSE**-Anweisungen und für die logische Verknüpfung von Daten eingesetzt. Logische Operatoren sind: **AND, OR** und **NOT.**

In der folgenden Diskussion sind **A** und **B** Verhältnisausdrücke mit dem Wert 1 (wahr) oder 0 (unwahr). Logische Operationen werden nach arithmetischen und Verhältnis-Operationen ausgeführt.

Operator	Beispiel	Bedeutung
NOT	NOT A	Wenn A wahr ist, ist NOT A unwahr
AND	A AND B	A AND B ist nur dann wahr, wenn A und B beide wahr sind. A AND B ist unwahr, wenn A oder B unwahr sind.
OR	A OR B	A OR B ist wahr, wenn A oder B oder beide wahr sind. A OR B ist unwahr, wenn A und B unwahr sind.

WAHRHEITS-TABELLEN

Die folgenden Tabellen werden Wahrheits-Tabellen genannt, Sie beschreiben das Ergebnis logischer Verknüpfungen für jede mögliche Kombination von A und B.

WAHRHEITSTABELLE NOT-FUNKTION

A	NOT A
WAHR	UNWAHR
UNWAHR	WAHR

WAHRHEITSTABELLE AND-FUNKTION

A	B	A AND B
WAHR	WAHR	WAHR
WAHR	UNWAHR	UNWAHR
UNWAHR	WAHR	UNWAHR
UNWAHR	UNWAHR	UNWAHR

WAHRHEITSTABELLE OR-FUNKTION

A	B	A OR B
WAHR	WAHR	WAHR
WAHR	UNWAHR	WAHR
UNWAHR	WAHR	WAHR
UNWAHR	UNWAHR	UNWAHR

Beispiel:

```
10 INPUT A, B, C
20 IF A = B AND B = C THEN PRINT "A = B = C"
30 IF (NOT A = B) OR (NOT B = C) THEN 50
40 END
50 PRINT "A = B = C IST UNWAHR"
60 END
   RUN
     ?    10
    ??     5
    ??     7
   A = B = C IST UNWAHR
```

KAPITEL 10

SCHLEIFEN-OPERATIONEN

— FOR. . .TO
— NEXT
— STEP

FOR ... TO ... NEXT ... STEP

Oft muß eine Gruppe Anweisungen für einen Prozeß mehrmals wiederholt werden. Mit der **FOR ... NEXT**-Schleifen-Anweisung braucht dieser Prozeß in **BASIC** nur einmal formuliert werden. Sollte beispielsweise der Kubus der Zahlen 1 bis 1Ø, dividiert durch drei, ermittelt werden, so kann dies mit dem folgenden Programm geschehen:

Beispiel:

```
1Ø FOR X = 1 TO 1Ø
2Ø PRINT X; X ↑ 3/3
3Ø NEXT X
4Ø END
   RUN
    1   .333333
    2   2.66667
    3   9.ØØØØ1
    4   21.3333
    5   41.6667
    6   72
    7   114.333
    8   17Ø.667
    9   243
    1Ø   333.333
```

Sie sehen, daß mit einer kurzen Sequenz der Computer Kubus und Division durch 3 der Zahlen 1 bis 1Ø errechnet hat. Mit der **FOR . . . TO**-Anweisung wurden die Zahlen 1 bis 1Ø erzeugt.

Mit jedem Schleifendurchlauf wird die Variable **X** automatisch um 1 erhöht. Eine Erhöhung um andere Werte kann mit **STEP** formuliert werden. Eine positive Zahl nach **STEP** erhöht, eine negative Zahl nach **STEP** erniedrigt. Es kann nach **STEP** eine Variable oder ein Ausdruck verwendet werden.

Beispiel:

```
10 FOR X = 1 TO 10 STEP 2
20 PRINT X; X ↑ 3/3
30 NEXT X
40 END
   RUN
```

Kubus und Division durch 3 werden jetzt für alle ungeraden Zahlen 1 bis 10 durchgeführt:

```
 1   .333333
 3   9.00001
 5   41.6667
 7   114.333
 9   243
```

Sie haben sicher schon festgestellt, daß jede Schleife mit einer **NEXT**-Anweisung abgeschlossen wird. Der Variablenname der **NEXT**-Anweisung muß der gleiche wie nach **FOR** sein. In diesem Falle ist er **X**. Er ist optional und kann entfallen.

Mit Hilfe der Schleifenanweisung lassen sich sehr einfach Tabellen erstellen, wie das nachfolgende Beispiel zeigt.

Beispiel:

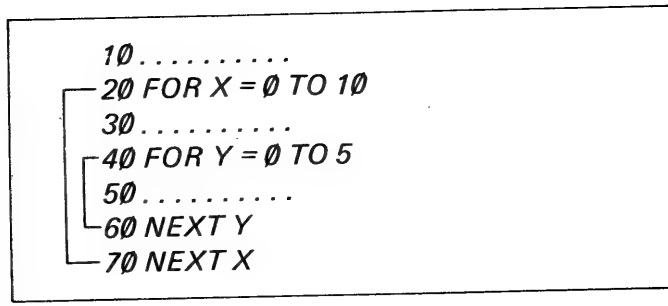
```
1Ø REM SINUS- COSINUS- TABELLE
2Ø PRINT "SIN (X)", "COS (X)"
3Ø FOR X = Ø TO 2 STEP Ø.5
4Ø PRINT SIN (X), COS (X)
5Ø NEXT X
6Ø END
   RUN

SIN (X)          COS (X)
Ø                1
Ø.479426         Ø.877582
Ø.841471         Ø.54Ø3Ø2
Ø.997475         Ø.Ø7Ø737 1
Ø.9Ø9298         Ø.416147
```

Soll innerhalb einer Schleife eine zweite Schleife formuliert werden, so ist darauf zu achten, daß die Schleifen sich nicht kreuzen. Im folgenden Beispiel wird dieser Fehler absichtlich gemacht. Nach **RUN** macht der Computer mit einer **NEXT WITHOUT FOR**- Fehlermeldung darauf aufmerksam.

Beispiel:

```
1Ø.....
2Ø FOR X = Ø TO 1Ø
3Ø.....
4Ø FOR Y = Ø TO 5
5Ø.....
6Ø NEXT X
7Ø NEXT Y
```

Korrekt wird so eine geschachtelte Schleife wie folgt formuliert:



Geschachtelte Schleifen sollten Sie entsprechend dieser Regel aufbauen.

KAPITEL 11

SUBROUTINEN

— GOSUB
— RETURN

GOSUB-RETURN

Jedes Programm hat einen Anfang und ein Ende. Es hat eine Struktur. Die Struktur entsteht durch die Aneinanderreihung kleinerer Anweisungs-Blöcke, die bestimmte Aufgaben ausführen. Einige Blöcke können gleich sein. Als Subroutine formuliert, braucht so ein Block nur einmal im Programm stehen und kann dann von jeder beliebigen Stelle im Programm zur Ausführung gebracht werden. Die Anweisungen dazu sind **GOSUB** und **RETURN**.

Beispiel:

```
10 PRINT "HALLO"
20 GOSUB 50
30 PRINT "AUF WIEDERSEHEN"
40 END
50 PRINT "WIE GEHT ES"
60 GOSUB 80
70 RETURN
80 PRINT "BIS SPAETER"
90 RETURN
   RUN
   HALLO
   WIE GEHT ES
   BIS SPAETER
   AUF WIEDERSEHEN
```

Die Zeilen werden in der Reihenfolge **10, 20, 50, 60, 80, 90, 70, 30, 40** bearbeitet. Sie können an diesem Beispiel sehen, wie die **GOSUB**-Anweisung arbeitet.

Die **GOSUB**-Anweisung veranlaßt den Computer, das Programm in der nach **GOSUB** notierten Zeilennummer fortzusetzen. Anders als bei der **GOTO**-Anweisung ist aber der Computer in der Lage, einen Rücksprung auf die der **GOSUB**-Zeile folgende Anweisungszeile auszuführen.

Der Computer bearbeitet die mit **GOSUB** aufgerufene Routine bis zur Erkennung der **RETURN**-Anweisung. Der Rücksprung zum Hauptprogramm wird dann ausgeführt.

Ein anderes Beispiel:

```
10 FOR X = 1 TO 5
20 GOSUB 60
30 PRINT X; S
40 NEXT X
50 END
60 S = 0
70 FOR J = 1 TO 4
80 S = S + J
90 NEXT J
100 RETURN
   RUN
   1     10
   2     10
   3     10
   4     10
   5     10
```

20 GOSUB 60 veranlaßt den Computer zu einem Sprung nach Zeile 60. Ab Zeile 60 bearbeitet er die Anweisungen, bis er auf die **RETURN**-Anweisung trifft. Der Rücksprung erfolgt dann nach Zeile 30, und die Schleife wird weiter bearbeitet.

Untersuchen Sie das vorstehende Programm, und finden Sie seine Funktion heraus!

KAPITEL 12

LISTEN UND TABELLEN

— FELDER (ARRAYS)
— DIM

FELDER UND DIM

Es gibt zwei Arten Variablen: einfache und Feld-Variablen. Bisher haben wir uns nur mit den einfachen Variablen befaßt. Betrachten wir nun die Feld-Variablen!

Ein Feld ist eine geordnete Datenliste. Sie ermöglicht eine einfache Handhabung von größeren Datenmengen. In diesen Listen können Zahlen oder Zeichenketten abgelegt werden. Um ein Feld einzurichten, ist dem Feld ein Name zu geben und es ist die Feldgröße zu bestimmen. Der Name ist ein Variablenname, z.B. **A$(5)**.

Der Feldname ist von einem Variablenname durch die in Klammern gesetzte Größenbeschreibung (Subskriptor) zu unterscheiden. Andere Feldnamen sind **A(2)**, **B(7)**, **G5$(7)**, usw.

Warum brauchen wir FELDER?

Nehmen wir als Beispiel an, Sie besitzen 1ØØ Bücher und wollen in Ihrem Computer ein Verzeichnis dieser Bücher anlegen. Wenn wir an jedes Buch einen Variablennamen vergeben, entsteht das folgende Programm:

Beispiel:

```
1Ø  A$ = "VOM WINDE VERWEHT"
2Ø  B$ = "OLIVER TWIST"
    .
    .
    .
    .
    .
    .
1ØØØ  Y$ = "BASIC PROGRAMMIERUNG"
```

Dieses Verfahren wäre sehr ineffizient und zeitaufwendig. Der bessere Weg wäre, ein Feld mit 1Ø Elementen anzulegen. **A$** wird dann der Feldname und 1Ø der Subskriptor. Betrachten wir das folgende Beispiel:

Beispiel:

```
1Ø REM BUECHERNAMEN
2Ø DIM A$(1ØØ)
3Ø FOR X = 1 TO 1ØØ
4Ø INPUT "BUCHNAME", A$(X)
5Ø NEXT X
   RUN
   BUCHNAME? VOM WINDE VERWEHT
   BUCHNAME?
```

Mit diesem Programm können Sie einfacher Ihre Buchnamen eingeben.

Bevor Sie mit einem Feld arbeiten können, muß mit einer **DIM**-Anweisung das Feld **DIM**-ensioniert werden. Im vorstehenden Beispiel lautet sie **DIM A$ (1ØØ)**. Der Computer reserviert daraufhin Speicherplatz für das Feld mit Namen **A$** mit 1Ø1 Feldelementen. Später können dann die im Feld abgelegten Daten sortiert, bearbeitet oder ausgegeben werden. Dieser Typ eines Feldes ist ein eindimensionales Feld und entspricht einer Liste.

Zweidimensionale Felder können mit zwei Subskriptoren ebenfalls eingerichtet werden. Zum Beispiel haben wir 5 Studenten, die je drei Examen abgelegt haben:

	EXAMEN (1)	EXAMEN (2)	EXAMEN (3)
STUDENT 1	50%	70%	90%
STUDENT 2	63	42	36
STUDENT 3	20	62	50
STUDENT 4	70	75	84
STUDENT 5	93	82	68

Die Ergebnisse können in einem zweidimensionalen Feld abgelegt werden.

Der Aufbau eines solchen Programmes beginnt also mit der Anweisung *DIM A (5, 3)*. 5 ist dann die Zahl der Studenten, und 3 ist die Anzahl der Examen. *A (4, 1)* enthält dann das Ergebnis des 1.Examens des 4.Studenten.

Dreidimensionale Felder werden mit *DIM A (3, 6, 2)* eingerichtet. Die Größe des Feldes in jeder Dimension ist nur durch den Speicher des Computers begrenzt. Fassen wir also zusammen:

A (X)	ist ein eindimensionales Feld,	
A (X, Y)	ist ein zweidimensionales Feld	und
A (X, Y, Z)	ist ein dreidimensionales Feld.	

Wird ein Feld angesprochen, ohne daß eine **DIM**-Anweisung voranging, setzt der Computer als Subskriptor automatisch für jede Dimension **10** ein.

Jedes Feld darf nur einmal im Programmverlauf dimensioniert werden.

KAPITEL 13

READ, DATA UND RESTORE

— READ
— DATA
— RESTORE

READ, DATA

Wird eine größere Datenmenge im Programm benötigt, so können diese Daten mit **DATA**-Anweisungen im Programm notiert werden und mit der **READ**-Anweisung verarbeitet werden. Der sonst notwendig werdende Gebrauch der **IMPUT**-Anweisung kann entfallen.

Beispiel:

```
10 DATA 10, 60, 70, 80, 90
20 READ A, B, C, D, E
30 PRINT A; B; C; D; E
   RUN
   10   60   70   80   90
```

Der **READ**-Anweisung folgt eine Liste mit durch Kommas getrennten Variablen.

Der **DATA**-Anweisung folgt eine Liste mit durch Kommas getrennten Ausdrücken. Es können numerische oder String-Ausdrücke verwendet werden. Die **READ**-Anweisung übernimmt also die Werte aus der **DATA**-Liste für ihre Variablen.

Bei jeder Variablenzuweisung mit **READ** wird der nächste Ausdruck aus der Dataliste entnommen. Wird **READ** ausgeführt, und ist kein Ausdruck in der **DATA**-Liste mehr vorhanden, erfolgt die Fehlermeldung **OUT OF DATA.**

RESTORE

Sollen die Werte der **DATA**-Liste später im Programm nochmals verwendet werden, so kann mit der **RESTORE**-Anweisung der Zugriff mit **READ** auf die **DATA**-Liste wieder auf das erste Element gerichtet werden.

Beispiel:

```
10 DATA 1, 3, 8, 9
20 READ A, B, D
30 RESTORE
40 READ X, Y
50 PRINT A; B
60 PRINT X; Y
70 END
   RUN
   1    3
   1    3
```

Die **RESTORE**-Anweisung macht mehrfachen Zugriff auf die **DATA**-Liste möglich.

Beispiel:

```
10 REM FINDE MITTELWERT
20 DATA 0.125, 3, 0.6, 7
30 DATA 23, 9.3, 25.2, 8
40 S = 0
50 FOR I = 1 TO 8
60 READ N
70 S = S + N
80 NEXT
90 A = S/8
100 PRINT A
   RUN
   9.5281
```

Benutzen wir nun die Examensergebnisse unserer 5 Studenten (Kapitel 12), um die **READ** und **DATA**-Anweisungen zu demonstrieren.

Beispiel:

```
10 CLS: DIM A(5, 3)
20 PRINT "ERGEBNIS": PRINT
30 PRINT TAB(8); "EX(1)   EX(2)   EX(3)"
40 PRINT
50 FOR J = 1 TO 5
60 PRINT "STUDENT"; J;
70 FOR I = 1 TO 3:READ A(J, I):PRINT A(J, I);:
   NEXT: PRINT
80 NEXT
90 END
100 DATA 50, 70, 90, 63, 42, 36, 20, 62, 50, 70, 75
110 DATA 84, 93, 82, 68
   RUN
```

KAPITEL 14

PEEK UND POKE

— PEEK
— POKE

PEEK (Adresse)

Die **PEEK**-Funktion liefert als Ergebnis den dezimalen Inhalt des Speichers, der mit der Adresse als Argument angesprochen wurde. Das Resultat ist stets im Bereich Ø bis 255.

Beispiel:

```
3Ø A = PEEK (28672)
```

Der Inhalt des Speichers 28672 wird der Variablen **A** zugewiesen.

POKE Adresse, Ausdruck

Die **POKE**-Funktion sendet einen Wert Ø bis 255 an den mit der Adresse angegebenen Speicher. Sie arbeitet komplementär zur **PEEK**-Funktion.

Beispiel:

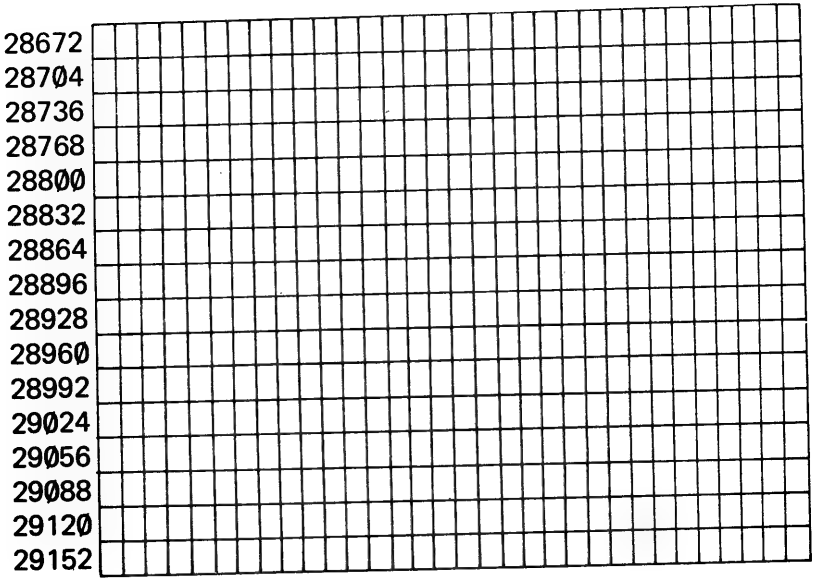
```
1Ø A = 1
2Ø POKE 29ØØØ, A
3Ø B = PEEK (29ØØØ)
4Ø PRINT B
   RUN
   1
```

Der Gebrauch des **POKE**-Kommandos sollte sehr vorsichtig erfolgen, da bei falschem Gebrauch das Programm gestört werden kann. Speichern Sie das Programm auf Kassette, bevor Sie **POKE**-Anweisungen ausführen lassen.

POKE-Anweisungen können nur für den Bereich des **RAM**-Speichers ausgeführt werden. (RAM = Schreib/Lese-Speicher). Dort werden das **BASIC**-Programm und die Werte der Variablen gespeichert. Der Bildschirmspeicher und der Tongenerator liegen ebenfalls im **RAM**-Bereich.

Es kann also möglich sein, das eine mit **POKE** angesprochene Adresse außerhalb des **RAM**-Bereiches liegt. Die Adresse für **POKE**- und **PEEK**-Anweisungen ist –32768 bis 32767.

Der Bildschirmspeicher, der **POKE**- und **PEEK**-Anweisungen zugänglich ist, beginnt hexadezimal bei 7000 (dezimal: 28672) und endet bei 71FF hexadezimal (dezimal: 29183). In der Grafik-Betriebsart ist der Bereich hexadezimal 7000 – 7FFF (dezimal 28672 – 36863).



Beispiel für die Anwendung der **POKE**-Anweisung:

```
10 CLS : SC = 28672
20 FOR I = 1 TO 9
30 READ A
40 POKE SC + I * 32, A
50 NEXT
60 GOTO 60
70 DATA 80, 79, 75, 69, 32, 80, 69, 69, 75
   RUN
```

Das Programm kann mit **CTRL** **BREAK** unterbrochen werden.

Beispiel für die Anwendung der **PEEK**-Funktion:

```
10 CLS : SC = 28672
20 PRINT "PEEK" : PRINT
30 FOR I = 0 TO 3
40 PRINT PEEK (SC + I);
50 NEXT
   RUN
```

Wenn sie **PEEK** und **POKE** in Verbindung mit dem Bildschirm-**RAM**-Speicher anwenden, beachten Sie:

Die Codes zur Bezeichnung der Character (Zeichen) entsprechen nicht dem üblichen ASCII-Code.

Orientieren Sie sich mit der folgenden Vergleichs-Tabelle.

(A)		(A)		(A)		(A)	
Ø	@	16	P	32		48	Ø
1	A	17	Q	33	!	49	1
2	B	18	R	34	"	5Ø	2
3	C	19	S	35	#	51	3
4	D	2Ø	T	36	$	52	4
5	E	21	U	37	%	53	5
6	F	22	V	38	&	54	6
7	G	23	W	39	'	55	7
8	H	24	X	4Ø	(	56	8
9	I	25	Y	41	)	57	9
1Ø	J	26	Z	42	*	58	:
11	K	27	[	43	+	59	;
12	L	28	\	44	,	6Ø	<
13	M	29	]	45	–	61	=
14	N	3Ø	↑	46	.	62	>
15	O	31	←	47	/	63	?

Mit (A) ist in der ersten Spalte der Character-Code bezeichnet. Die zweite Spalte enthält das Zeichen.

Der Code Ø bis 63 ergibt mit **POKE** in das Bildschirm-**RAM** das normale Zeichen. Wird dem Code ein Offset von 64 zuaddiert, werden mit **POKE** inverse Zeichen dargestellt (64 bis 127). Der Bereich 128 bis 255 ist in 8 Gruppen von Grafik-Zeichen aufgeteilt:

Gruppe	Code
1	128 – 143
2	144 – 159
3	16Ø – 175
4	176 – 191
5	192 – 2Ø7
6	2Ø8 – 223
7	224 – 239
8	24Ø – 255

(A)		(A)	
128		136	
129		137	
13Ø		138	
131		139	
132		14Ø	
133		141	
134		142	
135		143	

KAPITEL 15

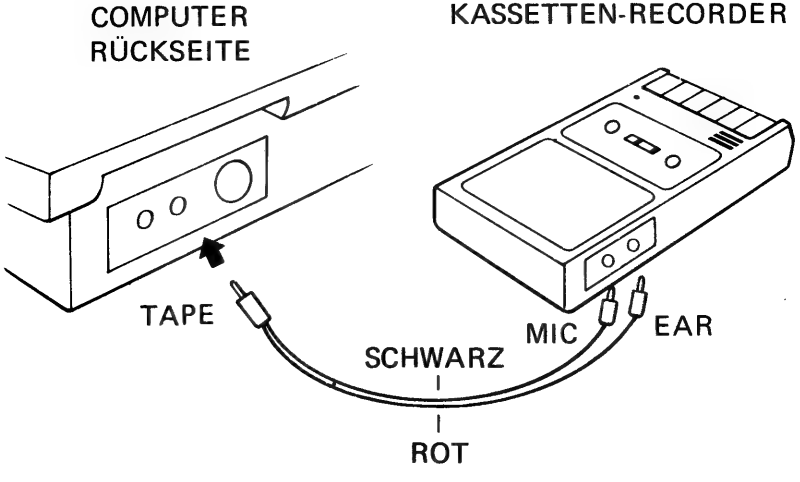
PROGRAMMSPEICHERUNG AUF KASSETTE

— VORBEREITUNG
— CLOAD
— VERIFY
— CSAVE
— CRUN
— PRINT#
— INPUT#

Bestimmt haben Sie inzwischen einige Programme entwickelt, die Sie häufiger benutzen wollen. Es ist sicherlich umständlich, sie jedesmal über die Tastatur einzugeben. Dieses Problem ist leicht gelöst. Speichern Sie Ihre Programme mit dem Kassetten-Recorder und laden Sie sie zum Gebrauch in den Computer-Speicher.

VORBEREITUNG

Sie brauchen einen normalen Kassetten-Recorder, Kassetten und das mitgelieferte Verbindungskabel. Schließen Sie den Recorder entsprechend dem Bild an.



Weiterhin müssen Sie sich mit drei Kommandos, nämlich **CSAVE, CLOAD** und **VERIFY**, vertraut machen.

Mit Ihrem Computer haben Sie eine Demonstrationskassette bekommen. Auf der einen Seite des Bandes ist ein Programm, die andere Seite ist leer. Versuchen Sie, dieses Programm in den Computer zu laden.

Jedes Programm auf der Kassette hat einen Namen (File-Name). Der Name ist ein Muß zum Schreiben auf Band, aber er ist nicht unbedingt notwendig zum Einlesen in den Computer oder zur Leseprobe (VERIFY) des aufgezeichneten Programmes. Der File-Name kann aus einem bis 16 Zeichen bestehen, wobei das erste ein Buchstabe sein muß, die anderen sind beliebig.

Zur besseren Verständigung: Lesen des Bandes bedeutet, ein Programm vom Band in den Computer zu überspielen. Schreiben auf Band ist das Überspielen eines Programmes vom Computer auf das Band.

VERIFY bedeutet, ein auf Band aufgezeichnetes Programm mit dem Programm im Computer zu vergleichen.

CLOAD "FILE NAME"

Zum Lesen eines Programmes:

1. Legen Sie das Band mit dem Programm in den Recorder ein.
2. Spulen Sie die Kassette bis zum Beginn des Programmes.
3. Geben Sie von der Tastatur *CLOAD "FILE NAME"*
   Drücken Sie noch nicht **RETURN** ! !
4. Drücken Sie die **PLAY** -Taste
5. Drücken Sie die **RETURN** -Taste
6. Der Computer gibt **'WAITING'** auf dem Bildschirm aus.
   Den Suchvorgang können Sie mit **CNTRL** **BREAK** abbrechen.
7. Ist ein gefundenes Programm nicht das gesuchte, erscheint *FOUND T: FILE NAME* auf dem Bildschirm.
8. Wird das gesuchte Programm gefunden und geladen, wird *LOADING T: FILE NAME* ausgegeben.
9. Wird die Meldung **READ** ausgegeben, drücken Sie die **STOP** – Taste des Recorders. Das Programm ist gelesen.

Angenommen, Sie haben auf Ihrem Band drei Programme mit den File Namen: **PROGRAMM 1, PROGRAMM 2** und **PROGRAMM 3.** Wenn Sie **PROGRAMM 3** einlesen, und starten Sie die Kasette am Bandanfang, so werden Sie nach *CLOAD "PROGRAMM 3"* **RETURN** auf dem Bildschirm finden:

```
CLOAD "PROGRAM 3"
WAITING
FOUND T: PROGRAM 1
FOUND T: PROGRAM 2
LOADING T: PROGRAM 3
READY
```

Wenn Sie die Kassette zum Beginn **PROGRAMM 3** spulen und dann das Programm einlesen, wird der Bildschirm folgendes zeigen:

```
CLOAD "PROGRAM 3"
WAITING
LOADING T: PROGRAM 3
READY
```

Anmerkung: 'T' steht für Textfile.

VERIFY "FILE NAME"

Um eine Aufzeichnung mit **VERIFY** zu prüfen, gehen Sie in folgender Weise vor:

1. Prüfen Sie mit **LIST**, ob das Programm im Computer vorhanden ist.
2. Schreiben Sie: *VERIFY "FILE NAME"*
   Drücken Sie noch nicht die **RETURN** -Taste!
3. Drücken Sie am Recorder die **PLAY** -Taste.
4. Drücken Sie **RETURN**.
   Der Cursor verschwindet, und der **VERIFY**-Vorgang beginnt.

Beispiel:

```
VERIFY "PROGRAM 2"
WAITING
FOUND T: PROGRAM 1
LOADING T: PROGRAM 2
VERIFY OK
READY
```

5. Die Meldung **VERIFY OK** zeigt, daß das Programm auf dem Band identisch mit dem im Computer ist.
6. Ergibt der Vergleich der Programme auf Band und im Speicher Abweichungen, so erscheint die Meldung: **VERIFY ERROR**. Das Programm sollte dann mit **CSAVE** noch einmal aufgezeichnet werden, und ein weiteter **VERIFY**-Vorgang sollte folgen.

Das Vorhandensein eines Programmes auf dem Band können Sie auch über den Tonkanal des Recorders feststellen.

CSAVE "FILE NAME"

Verwenden Sie zur Aufzeichnung von Programmen nur Kassetten guter Qualität. Sie können die Qualität Ihrer Aufzeichnung beeinflussen. Stellen Sie den Aussteuerungsregler Ihres Recorders auf den richtigen Wert. Er ist von Gerät zu Gerät verschieden. Den Klangregler stellen Sie auf **MAXIMUM.**

Schreiben Sie Ihr Programm nun auf Band:

1. Benutzen Sie für den ersten Versuch ein kurzes Programm. Längere Programme können Sie später auf Band schreiben.
2. Geben Sie das Kommando *CSAVE "FILE NAME"*.
   Es muß auf jeden Fall ein File-Name vergeben werden.
   Drücken Sie noch nicht die **RETURN** -Taste.
3. Setzen Sie eine Kassette guter Qualität in den Recorder.
4. Drücken Sie die **RECORD** und **PLAY** -Tasten.
5. Drücken Sie die **RETURN** -Taste.

6. Wenn der Cursor wieder sichtbar wird, ist die Aufzeichnung beendet.
7. Drücken Sie die **STOP** -Taste am Recorder.

Das Programm ist nun aufgezeichnet. Sicherheitshalber sollten Sie mit **VERIFY** die Aufzeichnung kontrollieren.

CRUN "FILE NAME"

Mit **CRUN** haben Sie eine weitere hervorragende Funktion. **CRUN** wird wie **CLOAD** gehandhabt, jedoch startet das Programm nach Beendigung des Ladevorganges automatisch.

Die vier Kommandos des Kassetten-Interface: **CLOAD, CSAVE, VERIFY** und **CRUN** helfen Ihnen, Programme auf Kassette zu speichern, sie zu testen, sie wieder zu laden und auszuführen. Achten Sie auf die Stellung des Aussteuerungsreglers. Dort liegt eine ständige Fehlerquelle!

Machen Sie sich jetzt mit zwei weiteren Kommandos des Kassetten-Interface vertraut:

PRINT# "FILE NAME", Variablen-Liste

Die Werte der in der Liste definierten Variablen werden auf Band aufgezeichnet.

INPUT# "FILE NAME", Variablen-Liste

Vom Band kommende Daten werden in die Variablen der Liste eingelesen.

Beispiel:

```
10 PRINT # "KAM", 1, 2, 3, 4, 5
   RUN
```

Die Daten 1 bis 5 werden als Daten-File "KAM" auf die Kassette aufgezeichnet. Der Recorder muß im **RECORDER**-Betrieb sein, bevor die Anweisung ausgeführt wird.

Beispiel:

```
10 INPUT # "KAM", A, B, C, D, E
20 PRINT A; B; C; D; E
   RUN
   FOUND D: KAM
   1  2  3  4  5
```

Die Daten vom Daten-File "KAM" werden vom Band eingelesen und den Variablen der Liste zugewiesen. Vor Ausführung der Anweisung sollte der Recorder auf **PLAY** geschaltet sein.

Anmerkung: D: steht für Daten-File.

KAPITEL 16

GRAFIK

— BETRIEBSARTEN
— GRAFIK-ZEICHEN
— INVERS-DARSTELLUNG
— SET
— RESET
— POINT

BETRIEBSARTEN

Der Bildschirm kann in zwei verschiedenen Betriebsarten betrieben werden. Nach dem Einschalten befindet er sich in der Betriebsart 'TEXT', **MODE (Ø)**. Dargestellt werden je 32 Zeichen in 16 Zeilen, bei Viertel-Grafik können 64 x 32 Punkte dargestellt werden. Text und Grafik können gemischt werden.

Der Anwender kann mit der Anweisung **MODE (1)** zur hochauflösenden Grafik umschalten. 128 x 64 Punkte können eingesetzt werden. In dieser Betriebsart lassen sich aussagekräftige Grafiken und Spiele erstellen.

Mit **MODE (Ø)** kann zum Textbetrieb zurückgeschaltet werden.

GRAFIK-ZEICHEN

16 Grafik-Character lassen sich mit **SHIFT** von der Tastatur eingeben. Mit diesen Zeichen lassen sich einfache Bilder und Grafiken darstellen.

Das folgende Beispiel zeigt Ihnen Möglichkeiten auf:

Beispiel:

```
10 CLS
20 FOR I = Ø TO 15
30 FOR J = Ø TO 1
40 FOR K = Ø TO 15
50 POKE 28672 + I * 32 + J * 16, K + 128
60 NEXT: NEXT: NEXT
70 GOTO 70
   RUN
```

In diesem Beispiel werden mit den Grafik-Charactern Muster aufgebaut. Das Programm benutzt den gesamten Grafik-Charactersatz. Sie können aber auch direkt von der Tastatur mit den entsprechenden Tasten dargestellt werden.

INVERS-DARSTELLUNG

Mit CNTRL REVERS werden alle Zeichen invers dargestellt. Die Rückschaltung zu den Standardzeichen kann mit CNTRL REVERS oder RETURN geschehen.

SET (X, Y)

In der Betriebsart **MODE (1)** setzt diese Funktion einen Punkt an der mit den Koordinaten **X** und **Y** bezeichneten Stelle des Bildschirms. Der Wert von **X** kann Ø bis 127 sein, der Wert für **Y** darf von Ø bis 63 definiert werden.

RESET (X, Y)

Diese Funktion schaltet einen mit der **SET**-Funktion gesetzten Punkt wieder aus. **X** und **Y** sind die Koordinaten zur Bestimmung des Punktes. Es gelten die gleichen Wertbereiche wie bei der **SET**-Funktion.

POINT (X, Y)

Diese Funktion ist wahr, wenn der mit den Koordinaten **X** und **Y** bezeichnete Punkt gesetzt ist, und unwahr, wenn der Punkt nicht gesetzt ist. Diese Funktion wird in **IF ... THEN ... ELSE**-Anweisungen eingesetzt.

Beispiel:

```
80 SET (40, 20) : IF POINT (40, 20) THEN
   PRINT "JA" ELSE PRINT "NEIN"
```

Hier folgen zwei Beispiele zum Gebrauch von **SET** und **POINT**:

Beispiel mit **SET** und **RESET**:

```
10 MODE (1)
20 FOR I = 0 TO 127
30 SET (I, 1/2)
40 NEXT
50 FOR I = 0 TO 127
60 RESET (I, 1/2)
70 NEXT
80 MODE (0)
```

Diese Beispiel plottet eine diagonale Linie auf den Bildschirm und löscht sie dann wieder. Am Ende des Programmes befindet sich der Computer wieder in der Text-Betriebsart.

Anmerkung: Soll der Computer in der Grafik-Betriebsart bleiben, ersetzen Sie Zeile 8Ø mit: *8Ø GOTO 8Ø.*

Das Programm kann dann mit CNTRL BREAK unterbrochen werden.

Beispiel zu **POINT:**

```
1Ø MODE (1): DIM A(1Ø)
2Ø FOR I = 1 TO 1Ø STEP 2
3Ø SET (I, I): NEXT
4Ø FOR I = 1 TO 1Ø
5Ø A(I) = POINT(I,I): NEXT
6Ø MODE (Ø)
7Ø FOR I = 1 TO 1Ø
8Ø PRINT A(I): NEXT
```

Dieses Programm plottet Punkte auf den Bildschirm. Die Anweisung **POINT** testet den Ort. Wenn der Ort mit einem Punkt besetzt ist, wird der Wert von A(I) zu 1, ist der Ort unbesetzt, wird A(I) zu Null. Am Ende des Programmes werden die Werte des Feldes A(I) ausgegeben.

KAPITEL 17

WEITERE KOMMANDOS UND INFORMATIONEN

— PRINT @ (PRINT AT)
— PRINT TAB
— PRINT USING
— INP
— OUT
— USR

PRINT @

Diese Anweisung lenkt die Ausgabe auf einen definierten Punkt des Bildschirmes. 512 mögliche Punkte existieren in dem 32 x 16 Punkte großen Bildschirmgitter. Die Anweisung wird in der Form

PRINT @ Position, Liste von Ausdrücken

gegeben. Die Position kann eine Zahl, eine Variable oder ein numerischer Ausdruck im Bereich Ø bis 511 sein.

Beispiel:

```
60 PRINT @ 60, 600;
```

Das Semikolon verhindert, das der Rest der Zeile im Bildschirm gelöscht wird.

PRINT TAB (Ausdruck)

Diese Anweisung wirkt wie der Tabulator einer Schreibmaschine. Der Cursor wird zu einem definierten Punkt der Zeile bewegt. Der Wert des Ausdrucks kann zwischen Ø und 255 liegen. Ist der Wert größer als 63, wird der Cursor zu einer Position entsprechend dem maximalen ganzzahligen Vielfachen von 64, bewegt.

Beispiel:

```
40 PRINT TAB (6); 1; TAB (20); 1
RUN
1                    1
```

PRINT USING String; Liste mit Ausdrücken

Diese Anweisung legt die Form der Ausgabe fest. Sie ist besonders gut für Listen und Tabellen geeignet.

Sie hat die Form:

```
PRINT USING String; Ausdruck
```

Der Ausdruck kann eine Variable oder eine Konstante sein, er kann numerisch sein oder eine Zeichenkette darstellen. Der Wert rechts vom Semikolon wird zur Ausgabe maskiert entsprechend dem Formatstring links vom Semikolon.

A) "!" Mit diesem Zeichen wird festgelegt, das nur das erste Zeichen eines Strings zur Ausgabe gelangt.

Beispiel:

```
10 A$ = "ASDF"
20 PRINT USING "!"; A$
RUN
A
```

B) "#" Dieses Zeichen stellt die Digit-Position einer Zahl dar. Soviel Digits, wie mit "#" definiert, werden ausgegeben. Hat eine Zahl weniger Digits, so wird sie rechtsbündig in das Ausgabefeld gestellt.

"." Der Dezimalpunkt kann an jede Position des Ausgabefeldes gebracht werden. Wenn der Formatstring ein Digit vor dem Dezimalpunkt erlaubt, wird das Digit auch ausgegeben (als Ø, wenn notwendig). Zahlen werden, wenn notwendig, gerundet.

Beispiel:

```
PRINT USING "##.##"; .78
Ø.78
```

C) "+" Ein Pluszeichen am Anfang oder Ende des Formatstrings bringt das Vorzeichen der Zahl (+/–) an den Anfang oder an das Ende der Zahl.
"–" Ein Minuszeichen am Ende des Formatstrings veranlaßt, das negative Zahlen mit einem nachgestellten Vorzeichen ausgegeben werden.

Beispiel:

```
PRINT USING "+##.##"; –68.95
–68.95
PRINT USING "##.##–"; –68.95
68.95–
```

D) "**" Ein doppelter Stern am Anfang des Formatstrings füllt vorangestellte Leerzeichen des numerischen Feldes mit Sternen auf. Mit ** kann auch die Position zweier weiterer Digits spezifiziert werden.

Beispiel:

```
PRINT USING "**#.#"; –Ø.9
*–Ø.9
```

E) "$$" Ein doppeltes Dollarzeichen veranlaßt die Ausgabe eines Dollarzeichens links vor jeder formatierten Zahl. Dieses Zeichen spezifiziert zwei weitere Digit-Positionen, eine davon wird von $ belegt. Das Exponentialformat kann mit $$ nicht angewandt werden, negative Zahlen können mit $$ nur bei rechts stehendem Zeichen ausgegeben werden.

Beispiel:

```
PRINT USING "$$###.##"; 456.78
$456.78
```

"**$" Dieses Zeichen am Anfang des Formatstrings kombiniert die Effekte beider Symbole. Vorangestellte Leerzeichen werden mit * gefüllt, und ein $-Zeichen wird vor der Zahl ausgegeben. **$ spezifiziert drei weitere Digits, eines davon für das Dollarzeichen.

F) "," Ein Komma links vor dem Dezimalpunkt veranlaßt die Ausgabe eines Kommas jeweils drei Digits links vom Punkt. Ein Komma am Ende des Formatstrings wird als Teil des Strings ausgegeben. Das Komma spezifiziert eine weitere Digitposition.

Beispiel:

```
PRINT USING "####,.##"; 1234.5
1,234.5Ø
```

G) "%" Wird eine Zahl, die größer ist als das festgelegte Feld, ausgegeben, wird vor der Zahl ein "%"-Zeichen ausgegeben. Wird das Ausgabefeld durch Aufrunden einer Zahl überschritten, so wird ebenfalls ein "%"-Zeichen vorgestellt.

Beispiel:

```
PRINT USING "##.##"; 111.22
%111.22
PRINT USING ".##"; .999
%1.ØØ
```

INP (I)

Diese Funktion liest ein Byte aus dem Port **I**. **I** muß im Bereich Ø bis 255 sein. **INP** ist die Komplementärfunktion zu **OUT**.

Beispiel:

```
100 A = INP (255)
```

OUT I, J

Diese Anweisung sendet ein Byte zu einem Ausgangsport. **I** und **J** müssen Ausdrücke im Bereich Ø bis 255 sein. **I** ist die Portnummer und **J** ist der Wert, der an den Port ausgegeben wird.

Beispiel:

```
100 OUT 32,100
```

USR (X)

Mit **USR (X)** wird eine Anwender-Maschinenroutine aufgerufen. Ihr kann das Argument **X** zur Bearbeitung übergeben werden. Die Subroutine kann vom Band oder durch **POKE** in den Speicher implementiert werden. Besondere Vorsicht sollte beim Aufruf dieser Anweisung geübt werden.

Beispiel:

```
110 A = USR (B/2)
```

KAPITEL 18

TÖNE UND MELODIEN

— SOUND
— MELODIEN

SOUND

Eine weitere interessante Eigenschaft des Computers ist die Möglichkeit zur Tonerzeugung. Hier ist ein Beispiel:

```
10 FOR I = 1 TO 8
20 READ X
30 SOUND X, 7
40 NEXT
50 DATA 16, 18, 20, 21, 23, 25, 27, 28
   RUN
```

Es werden acht auf der Tonleiter ansteigende Töne erzeugt. Die Variable **X** bestimmt die Frequenz, und die Konstante 7 ist die Tondauer.

Es können 31 verschiedene Töne mit 9 verschiedenen Tonlängen erzeugt werden. Die folgenden Tabellen zeigen die Zuordnung der Codes zu Frequenz und Dauer.

Ihrer Phantasie zum Komponieren ist also keine Grenze gesetzt.

TON-DAUER

Code	Note	Notenlänge
1		$\frac{1}{8}$
2		$\frac{1}{4}$
3		$\frac{3}{8}$
4		$\frac{1}{2}$
5		$\frac{3}{4}$
6		1
7		$1\frac{1}{2}$
8		2
9		3

FREQUENZ (Tonhöhe)

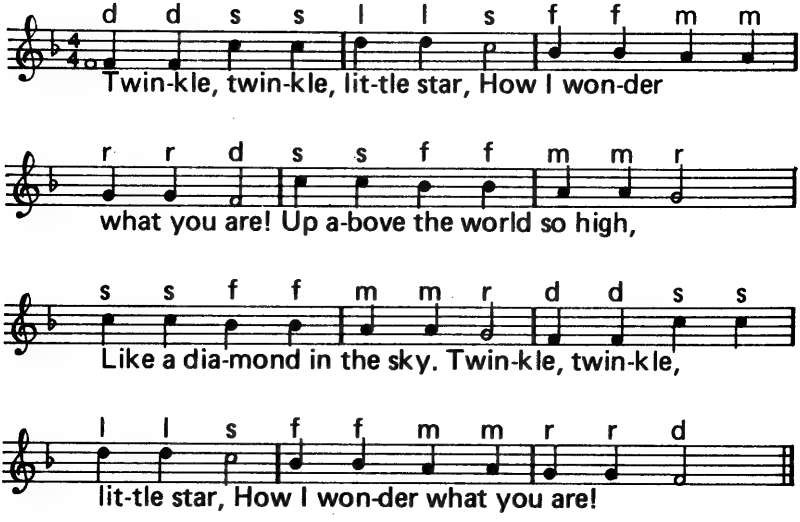
Code	Ton	Code	Ton
Ø	rest	16	C4
1	A2	17	C#4
2	A#2	18	D4
3	B2	19	D#4
4	C3	2Ø	E4
5	C#3	21	F4
6	D3	22	F#4
7	D#3	23	G4
8	E3	24	G#4
9	F	25	A4
1Ø	F#3	26	A#4
11	G3	27	B4
12	G#3	28	C5
13	A3	29	C#5
14	A#3	3Ø	D5
15	B3	31	D#5

MELODIEN

In dem jetzt folgenden Beispiel wird ein Musikstück zu einem Computerprogramm umgesetzt. Verfolgen Sie anhand der Tabelle die Umsetzung!

TWINKLE, TWINKLE, LITTLE STAR
Nursery Rhyme

Key F



TWINKLE, TWINKLE, LITTLE STAR

```
2 DATA 21,4,21,4,28,4,28,4,30,4,30,4,28,6,26,4,26,4,25,4
4 DATA 25,4,23,4,23,4,21,6,28,4,28,4,26,4,26,4,25,4,25,4,
       23,6
6 DATA 28,4,28,4,26,4,26,4,25,4,25,4,23,6,21,4,21,4,28,4,
       28,4
8 DATA 30,4,30,4,28,6,26,4,26,4,25,4,25,4,23,4,23,4,21,6
10 FOR I = 1 TO 42 : READ F, D : SOUND F, D : NEXT : END
```

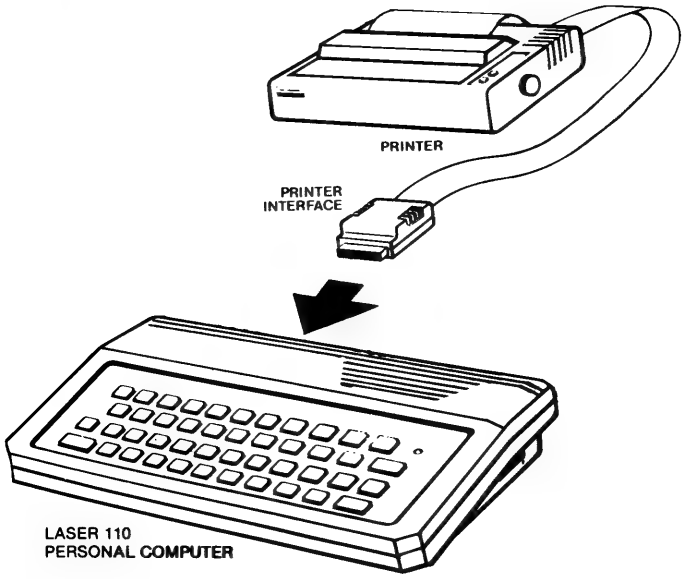
KAPITEL 19

DER DRUCKER

— LLIST
— LPRINT
— COPY

Um das Leistungsvermögen des Computers zu erweitern, kann ein Drucker angeschlossen werden. Er wird mit einem **PRINTER INTERFACE**-Modul angeschlossen. Mit diesem Interface bekommen Sie ein kleines Manual mit detaillierter Funktionsbeschreibung.

ANSCHLUSS DES DRUCKERS



Um erfolgreich mit dem Drucker zu arbeiten, machen Sie sich bitte mit den Anweisungen **LLIST, LPRINT** und **COPY** vertraut.

LLIST

Wie **LIST** auf den Bildschirm, gibt **LLIST** das Programmlisting auf den Drucker aus.

LPRINT

Wie **PRINT** auf den Bildschirm, arbeitet **LPRINT** zum Drucker.

COPY

Mit **COPY** wird ein Abzug des Bildschirmes an den Drucker ausgegeben. Es wird der Bildschirm kopiert.

Der Drucker kann mit den Tasten **CNTRL** **BREAK** angehalten werden.

Die **COPY**-Anweisung ist nur mit den Druckern **SEIKOSHA GP-100** und **GP-100A** möglich.

KAPITEL 20

HINWEISE ZUM EFFEKTIVEN PROGRAMMIEREN

HINWEISE ZUM EFFEKTIVEN PROGRAMMIEREN

Ein Programm ist dann als effektiv zu bezeichnen, wenn es so wenig wie möglich Speicherplatz belegt und wenn es in einer möglichst kurzen Zeit abgearbeitet wird.

Um diese zu erreichen, geben wir Ihnen nachfolgend einige Hinweise, die Sie für Ihre **BASIC**-Programme berücksichtigen sollten.

1) Für diesen Computer ist der Gebrauch von **'LET'** optional. Löschen Sie alle **'LET'**-Anweisungen aus Ihrem Programm und Sie sparen den entsprechenden Speicherplatz.
2) Die **'REM'**-Anweisung ist ein wichtiges Instrument zur ausführlichen Dokumentation Ihres Programmes. Sollten Sie Probleme mit dem Speicherplatz-Bedarf haben, so sorgen Sie für eine möglichst sparsame Anwendung von **'REM'**-Zeilen.
3) Mit Leerzeichen zwischen den Anweisungen wird ein **BASIC**-Programm lesbarer, aber auch sie belegen Speicherplatz. Verzichten Sie bei Zeit- oder speicherplatzaufwendigen Programmen auch auf die Leerzeichen.
   z.B.: 1Ø FOR I TO 1Ø wird: 1ØFOR I = 1 TO 1Ø.
4) Das **LASER**-Basic bietet Ihnen die Möglichkeit, mehrere Anweisungen in eine Zeile zu schreiben. Auch damit wird Speicherplatz gespart und so die Programmausführung beschleunigt. Z.B.: 1Ø FOR I = 1 TO 1Ø:PRINT "LASER BASIC": NEXT
5) Integer-Variablen belegen weniger Speicherraum als Fließkomma-Variablen. Benutzen Sie also, wann immer möglich, Integer-Variablen. Z.B.: Statt A = 5 schreiben Sie A% = 5.
6) Benutzen Sie Subroutinen, um von verschiedenen Stellen des Programmes gleiche Anweisungssequenzen zur Ausführung zu bringen.

7) Benutzen Sie so wenig wie möglich Klammern in von Ihnen formulierten Ausdrücken. Auch so können Sie Programmspeicherplatzs sparen.
8) Vereinfachen Sie Ausdrücke in Funktionen soweit wie möglich. Das Resultat kann dann schneller ermittelt werden. Zum Beispiel:

    1Ø A = 3 ↑2 + 17 –22/3: A = INT (A)

wird schneller errechnet, als

    1Ø A = INT (3 ↑2 +17 –22/3),

braucht aber mehr Speicherraum.
9) Bestimmen Sie mit der **DIM**-Anweisung die Größe des Feldes. Ohne **DIM**-Anweisung werden für jede Dimension 1Ø Elemente angelegt, auch wenn Sie nur 4 Elemente brauchen. Beachten Sie auch, daß für jede Dimension ein Element mit dem Subskript Ø angelegt wird. **DIM** A (3) dimensioniert ein eindimensionales Feld mit den Indizies Ø, 1, 2, 3.
10) Häufig wird eine Operation mit immer wieder anderen Parametern wiederholt. Um Anweisungen nicht ständig zu wiederholen und um Speicherplatz zu sparen, sollten Sie diese Parameter in **DATA**-Zeilen formulieren und mit Hilfe der **READ**-Anweisung bei Gebrauch einlesen.
11) Sollten Sie feststellen, daß für Stringvariablen nicht genügend Speicherplatz vorhanden ist, so benutzen Sie das **CLEAR**-Kommando mit einem größeren Wert.

ANHANG

— FEHLERMELDUNGEN
— ZEICHEN-CODE TABELLE
— LISTE DER BASIC-ANWEISUNGEN
— "QUICK-REFERENCE" DER BASIC-ANWEISUNGEN

"QUICK-REFERENCE" DER BASIC-ANWEISUNGEN

Hinweis:		
	arg	Argument (Konstante oder Variable)
	var	Variable
	cont	Konstante
	str	String (Zeichenkette)
	str arg	String-Konstante oder -Variable
	zn	Zeilennummer
	exp	Ausdruck

Nr.	Anweisung	Bedeutung und Schreibweise	Seite
1	ABS (arg)	– Absolutwert	
2	AND	– In Verhältnis- und logischen Ausdrücken – Hat den Wert Ø, wenn wahr, und 1, wenn unwahr	
3	ASC (str arg)	– ASCII-Wert in dezimal	
4	ATN (arg)	– Arcus-Tangens-Funktion – Ergebnis in rad – $-1ØE38 \leqslant arg \geqslant 1ØE38$	
5	CHR$ (arg)	– Zeichen wird durch ASCII-Code angegeben – $Ø \leqslant arg \geqslant 255$	
6	CLEAR arg	– Speicherplatz für Variable	
7	CLOAD "File Name"	– Lädt Programm von der Kassette – Die ersten 16 Zeichen des File-Namen werden gewertet	
8	CLS	– Bildschirm löschen	
9	CONT	– Fortsetzen der Programm-Abarbeitung	
1Ø	COPY	– Kopie des Bildschirmes auf dem Drucker anfertigen	
11	COS (arg)	– Cosinus-Funktion – Argument in rad – $-9999999 \leqslant arg \leqslant 9999999$	

12 CRUN "File name"
- Laden des Programmes von der Kassette und und anschließender Start
- Die ersten 16 Zeichen des File Namen werden gewertet

13 CSAVE "File Name"
- Speichern des Programmes auf Kassette
- Die ersten 16 Zeichen des File Namen werden gewertet

14 DATA cont, str cont, "str", . .
- Zuweisungswerte für die **READ**-Anweisung

15 DIM var (arg)
- Dimensionierung eines Feldes.
- Andere Form: **DIM** var$ (arg)
- Max. 3 Dimensionen: **DIM** var (arg 1, arg 2, arg 3)

16 ELSE
- Siehe **IF**

17 END
- Abschluß des Programmes

18 EXP (arg)
- Exponential-Funktion
- e = 2,71828
- $-10E38 \leqslant arg \leqslant 87$

19 FOR var = arg 1 TO arg 2 STEP arg 3
- Schleifen Anweisung
- arg 1 $\leqslant$ arg 2
- arg 3 $\neq 0$

20 GOSUB zn
- Sprung zur Subroutine beginnend bei Zeilennummer zn

21 GOTO zn
- Sprung zur Anweisung zn

22 IF exp 1 THEN exp 2 ELSE exp 3
- Bedingte Verzweigung
- exp 1, exp 2 und exp 3 können logische Ausdrücke sein
- exp 2 und exp 3 können Zeilennummern sein.

23 INKEY$
- Liest ein Zeichen von der Tastatur in eine Variable ein
- Normal: var$ = INKEY$

24 INP (arg)
- Liest Wert von im arg genannten Port ein
- Ø ⩽ arg ⩽ 255
- Normal: var = **INP** (arg)

25 INPUT str; varl (varl$),. . . .
- Gibt String aus und weist Wert von der Tastatur an var (var$)

26 INT (arg)
- Ermittelt den Integerteil einer Zahl
- −1ØE38 ⩽ arg ⩽ 1ØE38

27 LEFT$ (str arg, arg)
- Substring links von str arg mit arg Zeichen

28 LEN (str arg)
- Ermittelt Anzahl der Zeichen von str arg

29 LET var = numerischer Ausdruck
- Wertzuweisung an Variable
- Andere Form: **LET** var $ = str exp

3Ø LIST zn 1 – zn 2
- Anweisungsliste von zn 1 bis zn 2
- zn 2 ⩾ zn 1
- Andere Form: **LIST**

31 LLIST zn 1 – zn 2
- Anweisungsliste an Drucker

32 LPRINT arg 1 (str arg 1), . . .
- Wie **PRINT**, jedoch zum Drucker
- ; , können benutzt werden

33 LOG (arg)
- Ermittelt Natürlichen Logarithmus von arg
- arg > Ø

34 MID$ (str arg, arg 1, arg 2)
- Substring von str arg
- Länge von arg 1 arg 2 Zeichen

35 MODE (arg)
- Schaltet Display in Grafik- oder Textbetrieb
- arg = Ø oder 1

36 NEXT — Siehe **FOR**

37 NEW — Löscht alle Speicher und den Bildschirm

38 NOT — Logischer Ausdruck mit dem Wert 1, wenn wahr und Ø, wenn unwahr
— Kann Bestandteil von IF. . THEN sein

39 OR — Logischer Ausdruck mit dem Wert 1, wenn wahr und Ø, wenn unwahr
— Bestandteil von Verhältnis-Testanweisungen.

40 OUT (arg 1, arg 2)
— Sendet arg 2 an Ausgabeport arg 1

41 PEEK (arg) — Ergibt als Resultat den Inhalt der Speicherzelle arg

42 POINT (arg 1, arg 2)
— Ermittelt den Zustand eines Punktes der Bildschirmgrafik
— Ø ⩽ arg 1 ⩽ 127
— Ø ⩽ arg 2 ⩽ 63

43 POKE arg 1, arg 2
— Schreibt arg 2 in mit arg 1 bezeichnete Speicherstelle
— Ø ⩽ arg 2 ⩽ = 255
— –32768 ⩽ arg 1 ⩽ 32767
— Beachten Sie die Möglichkeit des System-Zusammenbruchs bei falscher Anwendung!!

44 PRINT arg 1 (str arg 1), ...
— Gibt arg oder str arg auf den Bildschirm aus
— ; und, können benutzt werden

45 PRINT TAB (arg 1) arg 2 oder str arg 2
— Wie **PRINT**
— Tabulator mit arg 1 Leerzeichen
— Ø ⩽ arg 1 ⩽ 255

46 PRINT USING str; arg 1, ...
- Wie **PRINT**
- Ausgabe im mit str spezifizierten Format

47 PRINT @ arg 1, arg 2, ...
- Wie **PRINT**
- Ausgabe von arg 2 ab der mit arg 1 bestimmten Stelle des Bildschirmes
- Ø ⩽ arg 1 ⩽ 511

48 READ var 1 (var 1 $), ...
- Wertzuweisung aus **DATA**-Liste an Variable

49 REM — Anmerkung in der Programmliste

50 RESET (arg 1, arg 2)
- Löscht einen Punkt der Bildschirmgrafik
- Ø ⩽ arg 1 ⩽ 127
- Ø ⩽ arg 2 ⩽ 63

51 RESTORE — Setzt Lesezeiger der **READ**-Anweisung auf den Anfang der **DATA**-Liste

52 RETURN — Rücksprunganweisung am Ende einer Subroutine

53 RIGHT$ (str arg 1, arg 2)
- Substring aus str arg 1, rechts beginnend arg, 2 Zeichen lang

54 RND (arg)
- Generiert Zufallszahl
- arg ⩾ Ø

55 RUN zn
- Start der Programm-Abarbeitung bei zn
- Andere Form: **RUN**

56 SET (arg 1, arg 2)
- Setzt einen Punkt der Bildschirmgrafik
- Ø ⩽ arg 1 ⩽ 127
- Ø ⩽ arg 2 ⩽ 63

57 SGN (arg) — Signum-Funktion

58 SIN (arg)
- Ermittelt den Sinus von arg
- arg in rad
- $-9999999 \leqslant \text{arg} \leqslant 9999999$

59 SOUND arg 1, arg 2
- Generiert Töne
- $0 \leqslant \text{arg } 1 \leqslant 31$
- $1 \leqslant \text{arg } 2 \leqslant 9$

60 SQR (arg)
- Wurzel von arg
- $\text{arg} \geqslant 0$

61 STEP — Siehe **FOR**

62 STOP — Halt der Programmbearbeitung

63 STR$ (arg) — Wandelt arg in str

64 TAN (arg)
- Tangensfunktion von arg
- arg in rad
- $-9999999 \leqslant \text{arg} \leqslant 9999999$

65 THEN — Siehe **IF**

66 TO — Siehe **FOR**

67 USR (arg)
- Ruft Objekt-Code Subroutine auf
- Rektor auf Startadresse:
  LSB in 30862
  MSB in 30863

68 VAL (str arg)
- Ermittelt den numerischen Wert des Strings
- str arg ist eine numerische Zeichenkette

69 VERIFY "File Name"
- Vergleicht Programm auf Kassette mit Programm im Speicher
- Die ersten 16 Zeichen von "File Name" werden gewertet

FEHLERMELDUNGEN DES BETRIEB-SYSTEMES

Nummer Fehlermeldung

1 BAD FILE DATA
Daten-Tvp der mit **PRINT#** auf Kassette geschriebenen Daten stimmt beim Einlesen mit **INPUT#** nicht mit dem zugewiesenen Variablen-Typ überein.

2 CAN'T CONT (Can't continue)
Ein Programm kann mit der **CONT**-Anweisung nicht fortgesetzt werden, weil . . . .
. . . ein Fehler im Programm die Unterbrechung verursacht hat.
. . . es nach der Unterbrechung verändert wurde.
. . . es nicht mehr existiert.
. . . es noch nicht gestartet wurde.

3 DISK COMMAND
Anweisungen aus dem Disk-**BASIC** wurden gegeben. Disk-**BASIC** existiert nicht.

4 DIVISION BY ZERO
Eine Division mit Null wurde im Verlauf der Lösung eines numerischen Ausdrucks versucht.
Wird im Verlauf der Lösung eines Ausdrucks ein Wert = Null zu einer negativen Basis erhoben, wird diese Fehlermeldung ebenfalls ausgegeben.

5 ILLEGAL DIRECT
Eine Anweisung, die nur innerhalb einer Programmliste zulässig ist, wurde direkt von der Tastatur eingegeben.

6 FUNKTION CODE (Illegal function call)
Ein Parameter einer mathematischen oder String-Funktion ist außerhalb des zulässigen Bereiches. Die Fehlermeldung

wird auch ausgegeben:

1. Subskriptor negativ oder unzulässig groß.
2. Das Argument der **LOG**-Anweisung ist = Ø oder negativ.
3. Das Argument der **SQR**-Funktion ist negativ.
4. Eine negative Mantisse mit einem nicht-integer Exponent tritt auf.
5. Aufruf der **USR**-Funktion mit fehlender Startadresse.
6. Allgemein unkorrektes Argument in vielen Anweisungen.

7 LOADING ERROR

Ein Problem hat sich beim Einlesen des Programmes von der Kassette ergeben. Das Programm ist nicht ausführbar.

8 MISSING OPERAND

Der Operand einer Anweisung ist nicht vorhanden.

Beispiel: *COLOR statt COLOR 5, 4*
MODE statt MODE (1)

9 NEXT WITHOUT FOR

Eine Variable in einer **NEXT**-Anweisung hat keinen Bezug zu Variablen in offenen **FOR**-Anweisungen, oder es existiert keine offene **FOR**-Anweisung.

10 OUT OF DATA

Für eine **READ**-Anweisung existiert kein ungelesner Wert in der **DATA**-Liste mehr.

11 OUT OF MEMORY

Diese Fehlermeldung wird ausgegeben, wenn ein Programm zu lang ist, zuviele **FOR. . .NEXT**-Schleifen oder **GOSUB**-Anweisungen aktiv sind oder wenn zu viele Variablen definiert wurden. Tief verschachtelte Ausdrücke werden ebenfalls mit dieser Meldung angezeigt.

12 OUT OF SPACE (Out of string space)
Der für Stringvariablen freie Speicherbereich ist vergeben. Mit **CLEAR** ist ein größerer Bereich zu reservieren.

13 OVERFLOW
Das Ergebnis einer Berechnung ist größer als der zulässige höchste Zahlenwert. Wird das Ergebnis kleiner als der kleinste zulässige Wert, so wird es zu Null gesetzt, und es wird keine Fehlermeldung ausgegeben.

14 REDIM'D ARRAY (Redimensioned array)
Zwei **DIM**-Anweisungen sind auf das gleiche Feld bezogen oder eine **DIM**-Anweisung wird gegeben, nachdem der Defaultwert 1Ø für dieses Feld vergeben wurde.

15 REDO
Während der Ausführung einer **INPUT**-Anweisung wird ein String einer numerischen Variable zugewiesen.

16 RET'N WITHOUT GOSUB
Eine **RETURN**-Anweisung wird gefunden, für die keine aktive **GOSUB**-Anweisung besteht.

17 FORMULA TOO COMPLEX (String formula too complex)
Ein String-Ausdruck ist zu lang oder zu komplex. Der Ausdruck sollte in zwei kleinere Ausdrücke aufgelöst werden.

18 STRING TOO LONG
Es wird der Versuch gemacht, eine Zeichenkette mit mehr als 255 Zeichen zu generieren.

19 BAD SUBSKRIPT (Subscript out of range)
Ein Feldelement wird mit einem Index angesprochen, der außerhalb des mit **DIM** festgelegten Bereiches liegt.
Ein Feldelement wird mit einer Anzahl Indizies

angesprochen, die nicht in der entsprechenden **DIM**-Anweisung festgelegt ist.

20 SYNTAX (Syntax error)
Eine Anweisungszeile oder ein direktes Kommando ist unkorrekt eingegeben worden. Falsche Zeichen, fehlende Klammern, falscher Gebrauch von ".,;:" usw. können die Fehlermeldung verursachen.

21 TYPE MISMATCH
Ein numerischer Wert wird einer Stringvariablen zugewiesen, oder umgekehrt wird einer numerischen Variablen eine Zeichenkette zugewiesen.
Ein Stringargument wird in einer numerischen Funktion formuliert, oder ein numerisches Argument wird in einer Stringfunktion benutzt.

22 UNDEF' D STATEMENT (Undefined line)
Eine nicht existierende Anweisungszeile wird mit **GOTO, GOSUB, IF. .THEN. .ELSE** oder **RUN** angesprochen.

23 VERIFY ERROR
Das Programm auf der Kassette ist nicht identisch mit dem im Speicher vorhandenen Programm.

- Zahlen werden mit einer Genauigkeit von 6 Stellen verarbeitet
- Der numerische Bereich: $10\ E{-}38 \leqslant N \leqslant 10\ E\ 38$
- Alle Anweisungen können vom Programm oder als Kommandos ausgeführt werden.

FUNKTIONEN:

1) Arithmetische Operatoren

   +, –, *, /, ↑

2) Vergleichs-Operatoren

   >, <, =, <=, >=, ><

3) Arithmetische Funktionen

   SQR – Wurzel
   INT – Integerteil einer Zahl
   RND – Zufallszahl
   ABS – Absolutwert
   SGN – Signaturfunktion
   COS – Cosinus
   SIN – Sinus
   EXP – e
   TAN – Tangens
   LOG – Natürlicher Logarithmus
   ATN – Arcus tangens

4) String Funktionen

   LEN – Länge des Strings
   STR$ – String des numerischen Arguments
   VAL – Numerischer Wert des Strings
   ASC – ASCII-Code
   CHR$ – Zeichen
   LEFT$ – Linke Zeichen
   RIGHT$ – Rechte Zeichen
   MID$ – Mittlere Zeichen
   INKEY$ – Zeichen von der Tastatur
   + – Verkettungsoperator

5) Logische Operatoren

AND — Verhältnis- und logische Ausdrücke setzen
OR — wahr = 1 und unwahr = ϕ
NOT

6) Grafik- und Tonfunktionen

CLS — Bildschirm löschen
COLOR — Farbe setzen
SOUND — Ton versch. Frequenz und Dauer erzeugen
MODE — Umschaltung alphanumerisch/Grafik
SET — Setzen eines Punktes der Bildschirmgrafik
RESET — Löschen eines Punktes der Bildschirmgrafik
POINT — Ermittelt, ob ein Punkt gesetzt oder gelöscht ist.

7) Programm — Anweisungen

DIM — Dimensionieren
STOP
END
GOTO
GOSUB
RETURN
FOR...TO...STEP
NEXT
REM
IF...THEN...ELSE
INPUT
PRINT
PRINT TAB
PRINT USING
PRINT @
LET
DATA
READ
RESTORE

8) Kommandos

LIST
RUN
NEW
CONT
CLOAD – Lese Programm vom Band
CSAVE – Schreibe Programm auf Band
CRUN – Lese Programm vom Band und starte es
CTRL/BREAK – Stop Programm
PRINT# – Schreibe Daten auf Band
INPUT# – Lese Daten vom Band
VERIFY – Vergleicht Programm vom Band mit Programm im Speicher

9) Andere Anweisungen

PEEK – Lese System-Speicherzelle
POKE – Schreibe in System-Speicherzelle
LPRINT – Ausgabe zum Drucker
LLIST – Programmlisting zum Drucker
INP – Lese Daten vom Port
OUT – Schreibe Daten zum Port
COPY – Kopie des Bildschirms zum Drucker
USR – Startet Anwender-Maschinen-Subroutine

POKE	ASCII	CHAR	POKE	ASCII	CHAR	POKE	ASCII	CHAR	POKE	ASCII	CHAR
32	32	SPACE	33	33	!	34	34	"	35	35	#
36	36	$	37	37	%	38	38	&	39	39	'
40	40	(	41	41	)	42	42	*	43	43	+
44	44	,	45	45	–	46	46	.	47	47	/
48	48	0	49	49	1	50	50	2	51	51	3
52	52	4	53	53	5	54	54	6	55	55	7
56	56	8	57	57	9	58	58	:	59	59	;
60	60	<	61	61	=	62	62	>	63	63	?
0	64	@	1	65	A	2	66	B	3	67	C
4	68	D	5	69	E	6	70	F	7	71	G
8	72	H	9	73	I	70	74	J	11	75	K
12	76	L	13	77	M	14	78	N	15	79	O
16	80	P	17	81	Q	18	82	R	19	83	S
20	84	T	21	85	U	22	86	V	23	87	W
24	88	X	25	89	Y	26	90	Z	27	91	[
28	92	\	29	93	]	30	94	^	31	95	←
32	96	SPACE	33	97	!	34	98	"	35	99	#
36	100	$	37	101	%	38	102	&	39	103	'
40	104	(	41	105	)	42	106	*	43	107	+
44	108	,	45	109	–	46	110	.	47	111	/
48	112	0	49	113	1	50	114	2	51	115	3
52	116	4	52	117	5	54	118	6	55	119	7
56	120	8	57	121	9	58	122	:	59	123	;
60	124	<	67	125	=	62	126	>	63	127	RUBOUT
64	192	@	65	193	A	66	194	B	67	195	C
68	196	D	69	197	E	70	198	F	71	199	G
72	200	H	73	201	I	74	202	J	75	203	K
76	204	L	77	205	M	76	206	N	79	207	O
80	208	P	81	209	Q	82	210	R	83	211	S
84	212	T	85	213	U	86	214	V	87	215	W
88	216	X	89	217	Y	90	218	Z	91	219	[
92	220	\	92	221	]	94	222	↑	95	223	←
96	224	□	97	225	!	98	226	"	99	227	#
100	228	$	101	229	%	102	230	&	103	231	'
104	232	(	105	233	)	106	234	*	107	235	+
108	236	,	109	237	–	110	238	.	111	239	/
112	240	0	113	241	1	114	242	2	115	243	3
116	244	4	117	245	5	118	246	6	119	247	7
118	248	8	121	249	9	122	250	:	123	251	;
124	252	<	125	253	=	126	254	>	127	255	?

KONTROLL-KODE TABELLE

ASCII	FUNKTION	ASCII	FUNKTION	ASCII	FUNKTION	ASCII	FUNKTION
Ø		1		2		3	
4		5		6		7	
8	CURSOR LINKS	9	CURSOR RECHTS	1Ø	CURSOR ABW.	11	
12		13	CR	14		15	
16		17		18		19	
2Ø		21	INSERT	22		23	
24	CURSOR LINKS	25	CURSOR RECHTS	26		27	CURSOR AUFW.
28	CURSOR HOME	29		3Ø		31	CLEAR SCREEN

LASER™
PERSONAL COMPUTER 110

BASIC
PROGRAMMIERHANDBUCH

MADE IN HONG KONG
91-0123-04